MATHEMATICAL CENTRE TRACTS 29

J. VERHOEFF

ERROR DETECTING DECIMAL CODES

MATHEMATISCH CENTRUM AMSTERDAM 1975

AMS(MOS) subject classification scheme (1970): 94A10

ACM -Computing Review- category: 5.6

ISBN 90 6196 042 8

1st printing 1969
2nd printing 1975

PREFACE

A book is written for its readers, as the rumour goes. During the writing the author begins to form an image of these unknown people. They are supposed to know everything which the author is not willing to explain. In the underlying case it means that they know what a group is and that they know about computers and parity checks. On the other hand, they are not supposed to know much about group theory, so that several elementary concepts have to be explained. Chapter 0 contains a, philosophically tinted, discussion, aimed at clarifying the mental attitude of the author towards the coding problems. It is thought to be understandable to almost everyone. Chapter 1, on the contrary, is more specialistic and gives some upper bounds for the size of error detecting codes. It it the most unfinished part, but as such it reflects the state of art, since very often the limits are the last to be known to science.

The reader who is only eager to know what codes are available to him, can safely skip chapter 1. For him chapter 2 offers a survey of what is available. For his convenience the new codes, which are constructed in the chapters 3, 4, and 5, are also reviewed. The mathematical level of this chapter is, in view of the supposed level of his mathematical erudition, kept as elementary as possible. Even if the group concept is occasionally mentioned, it is only meant for those who can appreciate it, whereas the others can ignore it without losing the thread of the argumentation. The application eager reader will probably stop here, since the next chapters deal primarily with the methodology of finding codes with prescribed properties. That is why in these chapters the mathematical level is higher in the sense that more complicated notations and arguments are used. In chapter 3 the possibilities for codes based on the addition modulo 10 are screened, whereas in chapter 4 the analogous problem for codes based on the dihedral group of the order 10, is solved. Finally, chapter 5 deals with the so-called biquinary codes, culminating in a new and quite remarkable one.
For insiders, it may be pointed out that chapters 4 and 5 give pure decimal codes, which detect all transcription errors and all transpositions of adjacent symbols. This refutes the non-existence "proof"

occurring in the literature. The author believes that the codes explained in chapter 4 provide the first practical application of the dihedral group. This would illustrate the old saying that all beautiful mathematics will find an application, sooner or later.

For the sake of completeness, the bibliography also refers to relevant literature not mentioned in the text.

The author wishes to express his gratitude to the Delft University of Technology and the Mathematical Centre at Amsterdam, for putting their computer facilities at his disposal, and to the Amsterdam Municipal Clearing Office and the Netherlands Postal- Check and Giro Service, for confiding him in their error statistics and for their stimulation. His thanks also go to Mr. A. Benard and Dr. A.D. Colenbrander for allowing him to disclose their codes. He is also much indebted to the professors Dr.Ir. A. van Wijngaarden and Dr. W. Peremans, who carefully read the manuscript, for their many valuable suggestions.

It is impossible to mention everybody who has stimulated or helped the prepration of this book, but an exception will be made for the staff of the Mathematical Centre, who has to be complimented for the record speed with which the book was reproduced.

Contents

Chapter O. Introduction.

0.0. Coding, codes and notation.

A general word like code, is one of the hard working words, in the sense of Humpty Dumpty. Originally code referred to a law, written or not. In cryptology the word code is used in contradistinction to a cipher. By a code a system of substitution is meant in which many words, phrases or syllables are replaced by code words or code numbers. The word cipher refers to a system in which the individual letters are worked upon.
The commercial codes of the late twenties were used to cut down the costs of cablegrams. The military codes have secrecy as main purpose. Nowadays codes are widely used in the theory and practice of switching circuits, culminating in the design and use of computers.
In this monograph coding is understood to be a mapping of an arbitrary set into a set of mathematical entities. The first set is often a set of tangible objects, persons or concepts, whereas the second set mostly consists of symbols or strings of symbols. The structural formulae of organic chemistry form an example of the application of other mathematical entities than strings of symbols.
It is quite essential that the mapping is one to one, since the basic idea is, to use the abstract entities as names for the elements of the first set. In practical cases the main difficulty lies in the definition of the mapping. One can hardly attach the abstract entities to the objects or persons, be it that only the persons might object. This is, of course, the denotation problem, which is solved, more or less, by the use of tokens. Tokens are physical representations of symbols. For every symbol, there is a whole class of different tokens, which are commonly understood to stand for the same symbol. Examples are all types of "three's" in all kinds of colours, print, written or spoken, including the less generally agreed upon way to represent a "3" in a computer. The borderlines of these classes are sometimes dangerously vague. The choice of most tokens, which was made historically, would nowadays be called a very poor job of system design, as everyone involved in character recognition will concede. But it is too late for a change, all trials to introduce a new alphabet will be utopic. The world will have to live with the old one. Returning to

the definition of the mapping, it will be clear that one can attach to objects one or more tokens, as a label. These tokens represent the symbols on which the object has been mapped by the coding. Branding cattle may be one form and engraving a serial number in guns or engines is another way of explicit labeling. The labeling may have a dual purpose, since it may be done in order to make identical objects different. On the other hand, the branding of the cattle may be done to establish the ownership. Anyhow such an effective, but crude way to define a coding is impossible if the "objects" to be coded are concepts. For people the method may theoretically be feasible, but is hardly advisable, especially if these people are customers. The customary procedure is then to make some list, in which series of tokens, representing the code, are linked with a verbal description of the coded objects. Such a list is called a code book or catalogue. Actually the situation is rather tricky, since it might be said, that such a description itself is (a notation for) a code. Hence the question would remain how to define the latter code. Since a verbal description seldom really characterizes the object, it may be questioned whether such a description is a code. This does not make the situation any better. In fact, the descriptions use as a rule, all kinds of contexts, written or not, to help define the objects. Often it is supposed to be clear that the object is one of a known (how?) class. This type of problems is of course inherent to all (succesful) communication. Strictly speaking, communication is essentially impossible, but it sometimes works. It is merely a matter of success and efficiency how far one has to go with refining the descriptions. Parenthetically it may be remarked that the characterization of persons by fingerprints or sets of measures, may be very practical, but theoretically the system is never foolproof, and that not alone because of the fingerless people. There is a difficulty for every solution. Summarizing the one to one mapping of an arbitrary set into a set of mathematical entities is called coding. The second set is then called a code. Notation is a physical representation of the second set by means of tokens. These tokens fall apart into classes of equivalent ones, each representing the same mathematical symbol. The equivalence

is based on a common understanding and is as such a potential source of confusion. To help to avoid or at least detect this confusion is the aim of the following chapters.
An important point is that the common understanding of the tokens is some kind of social phenomena. It is as such amenable to a study, which will of course be of a statistical nature. It is conceivable to measure the degree of intersubjectivity by controlled experiments. One could let may people write a "3" and one could then measure how often, other or the same, people recognize it correctly. It might turn out that a "3" is a better token than say a "5". It is rather difficult, if not impossible, to get unbiased information on this recognition problem. The known error statistics on codes show a certain onesidedness (see 36) in the sense that, e.g., a "q" can be easily mistaken for a "g", but seldom a "g" becomes a "q". This may very well be due to the relative frequencies of use of the various tokens.

0.1 The application of decimal codes.

The use of the decimals is rooted in tradition. Unlike the binary codes, there is no intrinsic reason for its use. It is just because people are used to it, that the decimal system is so important. It is therefore not surprising, that the decimal codes are mostly handled, at least partly by human beings. The same holds in a way for alphabetic codes. For mnemotechnic reasons, it was believed in the past, that codes for human use, should be of the alphabetic or alpha-numeric type. Car licence numbers and telephone numbers in various countries are relics of this belief. In the present time where the human use and the machine handling gets mostly combined, the decimal codes are getting more popular. An other reason may be that recent studies indicate that the alphabetic characters are more error prone than the decimals (see 36).
As said before, the code words are intended to be used as names for the things for which they stand. A name is needed if a reference is to be made to something. Such a reference will be called a mutation. As a rule, the mutations are part of a process, say an administrative one. For the process the code words serve as input. The primary reason for the use of a code, rather than the natural language, is the efficiency. The fact that a code is unambiguous, is not a good argument, since that effect

could also be obtained by properly extending the natural names or descriptions, so that here again the efficiency is the basic motive In data processing systems it is customary that the various inputs are unrelated and come from many origins. Thus the mutations converge into the system. The coding is often done in the periphery, by the users or customers and is therefore largely outside the control of the system. As a consequence the system has to cope with the errors made. The redundant codes, which will be dealt with in the next section, serve as a defense of the system against these errors. To be sure, some of the errors are caused by human operators incorporated in the system, for the preparation of the machine readable records. This operation is of course under the systems responsability, and error prevention should be practiced anyhow. The redundant coding is in fact a burden for this preparatory operation which causes some authors to reject redundant coding altogether (see 6). But there is a tendency to push the preparation of the machine readable record back to the user. Optical readers, dials and on-line input stations are some of the means to that end. This self service eliminates the bottleneck of the punching and the like and as a rule deminishes the waiting time since batch forming may be avoided. This greatly widens the applicability range of the modern data processing systems. It also will, in the opinion of the author, make the use of redundant codes more urgent. In terms of the information theory it can be said, that a large number of channels converge into the system. The letters of the alphabet used in each channel, are the words of the code. The alphabets tend to be very large and the rate by which the letters are generated per channel will be very low. The channels are not noiseless. The noise is mostly caused by human factors. Mathematically, the noise is defined by the transition probabilities $p(x,y)$, where $p(x,y)$ is the chance that the code word x is received by the system as y. The nature of this noise will be the subject of section 0.4.

0.2. Redundant codes.

A code is called redundant as soon as the mapping of the coded set does not cover the code. This redundancy can be more or less accidental, because the code happens to have more words than necessary for the set

to be coded. In most applications, this will be the case, since populations, customers, inventories etc. do not tend to come in powers of 10, like the decimal codes do. The redundancy can also occur intentionally and sometimes temporarily, when the code is chosen purposely too large for a growing stock or population. Though the control of this natural redundancy is worthwhile, the main topic of the following chapters will be that of the artificial redundancy. The latter form of redundancy is obtained by admitting only a subset of the code for use. Strictly speaking only the admitted subset itself is the code. The words outside this subset are sometimes called improper or forbidden code words. This terminology has the same inconsistency, not at all unusual in mathematics, which adorns expressions like the burnt down house.

The code is some sort of intermediary between the real thing and the denotation. It is therefore typical that the mathematical properties of the code are sometimes desirable for the sake of the coded objects and sometimes for the sake of the notation. Hierarchical codes, like the U.D.C. are examples of the first kind. The teletype code examplifies the second kind. It is a 5 dimensional binary code since the teletype uses 5 channels. The physical representation, with the 2 states, hole or no hole, is of course a notation.

As will be seen later on in the section on errors, it is advantageous to introduce a topology or metric in the code, just to be able to describe the errors which result from the deficiencies of the notation. These errors provide the criteria for the selection of the subset which is to form the redundant code. Apart from their use in the struggle against errors the codes are of interest as a mathematical object of study.

The redundancy can be measured as follows. Let U be the set of potential code words and let C be the selected subset containing the proper code words. The fraction $1-|C|/|U|$ is a measure for the redundancy. The parity check would thus yield a code with a redundancy of 50%. By taking the base 2 logarithm the redundancy is measured in bits. The parity check has of course a redundancy of 1 bit. If the code words consist of m-ary digits, then the base m logarithm gives the redundancy in

in m-ary digits.

0.3. The detection of errors by redundant codes.

As stressed before, the main objective of coding is the increase of efficiency in handling the coded data. This holds for human processes as well as for automatic processes. Especially for the latter type it is important that the codes lend themselves for standardized notations. This is merely another aspect of the efficiency, but it shows again that the codes are not made for the peculiarities of the coded objects alone. In this machine age, people sometimes have to adapt themselves to the machine. The reason is, may be, that the machines do such a tremendous amount of data handling, that the pay-off from the efficiency in the machine part is more important. This may very well change when the cost per operation goes further down.
The drawback of the increased efficiency is that errors tend to be more dangerous : an error in a natural name, does not always produce another name, but a number is always changed into another number. One might also say that the numbers are all alike or that the names satisfy certain syntactical or even semantic rules. As not all letter combinations are used as names, it may be said that the names are highly redundant. In fact the set of names forms a, perhaps ill defined, redundant code. As soon as there is redundancy in a code there is a chance that an erroneous code word does not correspond with an object. Let A be a set of coded objects and let C be a selected subset of the set U of code words. Let c be a mapping of A in C, hence for all $a \in A$ it holds that $c(a) \in C$ and $c(A) \in C$. If $x=c(a)$ with $a \in A$, and if an error changes x into y, then there are three possibilities:

i) $y \in c(A)$; ii) $y \notin c(A)$ and $y \in C$ and iii) $y \notin C$.

In the first case the error is fatal in the sense that a false mutation may be made. In practice it will be often possible to detect the error from the context, written or not, like in the case that Granny was drafted for the commandoes. As a matter of fact that is how one knows about the errors anyhow, for as a rule there is a feed-back into the system from complaining customers or victims. Sometimes however the complaints are too late to undo the fatal consequences.

This first case is obviously very undesirable.
The second case is less harmful, since it cannot result into a wrong mutation. The code word y simply does not correspond to an object. It may cause some nuisance, since as a rule it will be detected during the processing. This may be by a mailman looking for a non-existent house. The automatic detection of these unused code words is under certain conditions possible. A simple example is a code of which it is known that only the first n words are used. It is of course necessary that the code words are ordered, say lexicographically. In the Dutch population registration number a more sophisticated method has been applied.
The third case is the most important one from a theoretical point of view. The error may in that case be detected without any knowledge of the use of the code. Especially if C is defined by an algorithm, it is possible to detect the error automatically. It should be noted that the second class of errors can always be converted into the third class by the application of a table look-up procedure. The art of making error detecting codes consists of two things; the first one is to select the set C in such a way that the most likely errors always belong to the third class and the second one is to define such a set C by means of a simple criterion, which lends itself to an easy technical implementation. The latter requirement is a matter of economy and as soon as the table look-up procedure is feasible, the requirement looses its importance. In general it is true that, when the memories get cheaper, algorithms can be (economically) replaced by table look-up. All these technical considerations are very much dependent on the state of technology. When the computers get better at parallel processing, the algorithm might again be more economic.
There is a tendency nowadays to adapt the machine to the human being, rather than the other way around. High level programming languages are also evidence of that tendency. When the computers are learning the natural language, the coding problems will be change, but not disappear. To be in vogue, the question of optimal error detecting codes should be considered. It would have to be a code, which detects more errors than any other code with the same redundancy. This property would

clearly be dependent on the frequency of use of the various code words and of the distribution of the errors. The optimalization problem is not very meaningful, for even if the costs of detected and undetected errors were known, they are bound to change in the course of time. Moreover, the distribution of the errors is usually unknown at the moment that the code has to be chosen. Furthermore, these statistical qualities may be expected to change during the existence of the system. Finally, though pure mathematically speaking there is no problem at all, since there is "only" a finite number of possibilities, from a practical point of view the problem is probably just as practical as the differential equations governing the universe. A situation like this provides for the mathematician a rich hunting ground for nice problems (see chapters 3 and 4).

0.4. The efficiency of a code.

The efficiency of a code depends on the frequency with which the code words occur. Let p denote this frequency distribution. It is well known from information theory, that it is always possible to encode a source with entropy $-\sum_{x \in C} p(x)\lg(p(x))$, so that the average length of the code words is equal to the entropy. Unfortunately, this theorem is rather sterile in cases where the coding lies outside the control of the system. It is all right in simple cases like the following one. Let a code consist of 4 words with mutation frequencies of 1/2; 1/4; 1/8 and 1/8 respectively. The entropy is in that case $-(2^{-1}\lg 2^{-1}+4^{-1}\lg 4^{-1}+8^{-1}\lg 8^{-1}+8^{-1}\lg 8^{-1})=7/4$. Encoding the words with 0, 10, 110 and 111 respectively, gives exactly the average of 175 bit per 100 mutations. If, however, the frequencies are not so civilized, then the proof of the theorem hinges on the trick of making pairs. The pairs of words form a larger set, within a "more uniform" statistical distribution of use. E.g., let A,B,C, and D be 4 words with a mutation frequency of 60%; 30%; 5% and 5% respectively. The above mentioned code would score an average of 1.5 bit per mutation whereas the entropy is roughly 1.4. Coding the pairs as follows gives an average of 1.43 bit per word.

AA=0	AC=11011	BC=110101	CC=110100110
AB=100	CA=11110	CB=111010	CD=110100111
BA=101	AD=11111	BD=111011	DC=110100100
BB=1100	DA=11100	DB=1101000	DD=110100101

Now in data processing systems in which the code words are generated independently at various locations, before they are channeled into the system, only the first approximation seems to be possible. This is not quite true, since one could do the pairing at the various sources, but it would require a code book of all the pairs. Now just imagine a bank publishing the book of the code numbers of all the pairs of account holders, not to speak of the letter headings of the customers.

The codes mentioned above are intended for use in a binary channel, where all the words are linked together and thus have to be separable afterwards. If there is a natural separation between the different words, it is possible to obtain a bigger gain in efficiency. Consider the second example again. Now the first approximation may be taken as A → 0; B → 1; C → 00; D → 01. It would give an average of 1.1 bit per mutation. The second approximation, given below, would require only 0.8 bit as an average.

AA → 0	AC → 10	BC → 010	CC → 110
AB → 1	CB → 11	CB → 011	CD → 111
BA → 00	AD → 000	BD → 100	DC → 0000
BB → 01	DA → 001	DB → 101	DD → 0001

It should be noted that even in the case of a uniform distribution, this type of coding would give a gain in efficiency.

All these considerations are probably of little practical value, since the distribution of use will only become available after the code has been used for some time. The cost of recoding will mostly outweigh the possible gains. In practice the distribution can be extremely skew, like in banking operations where often less than 1% of the accounts draw more than 50% of the mutations. The short code numbers are however often more correlated with the old clients rather than with the mutation getters.

In population register systems the distribution will probably be closer to uniform.

0.5 The error statistics.

The double distribution f(x,y) contains a host of information, which is usually not available. This is certainly so during the design stage of the system. Later on, when the system is operating, that information will become available. It is then still useful in the struggle against the errors. Suppose that in the administrative system of a bank, a certain error probability f(a,b) gets high, then this may be an indication of a systematic error, which lies possibly outside the system. It might be a misprint in somebodies account number, as indicated on his bills. These kinds of errors are only of local interest. In general however one will be interested in deducing principles, like the law that most errors are in one digit only. The knowledge of the double distribution is therefore more qualitative than quantitative. But, the vaguer the knowledge the broader the applicability.
The error samples, as found in existing systems, will be biassed if the distribution of the mutations is not uniform, i.e., virtually always. The following error types have been observed both in the literature on the subject (see 2, 26, 28) and from samples put available to the author by the courtesy of the Dutch Postal Clearing House and the Clearing House of the Amsterdam Municipality.
1) The single errors, also called transcription errors: These errors affect only one digit of the code word. It is by far the largest class of errors in all known cases. Its frequency ranges from 60 to 95%. Little is known about the distribution within this class. Both clearing house samples suggest that the right hand side of the code word is more vulnerable for errors. This might be caused by the fact that the numbers are written and punched from the left to the right, so that the right most digits have to be memorized longer in the short term memory of the writing or punching being. The number systems in which this was observed are of the non-fixed length type. This implies that the last position is never void contrary to the other positions. The transition probabilities of the decimals are indeed depending on the decimals, but there are few very low ones. (See the tables at of the next section).
W. Ulrich (49) introduced the concept of the restricted single error,

which is a single error with the restriction that the difference between the correct and the incorrect digit is one unit. This concept gives rise to elegant generalizations of known binary codes. It is conceivable that certain technical implementations of calculators, like the ones using pulse trains to represent the digits, lead to this error type. But there is no evidence in the error statistics that this type is of special interest.

2) The double errors. These errors affect two digits. The frequency ranges from 10 to 20%. The vast majority concerns adjacent digits, i.e., digits with adjacent positions. This is of course an indication that the two errors are not independent. The double errors are subdivided into:

2.1) The transpositions. Most of the adjacent double errors are of the form ab→ba. This error type is called a transposition. It will always be understood that the digits are adjacent if in the following chapters the term transposition is used.

The transpositions are a notorious error type of a typical human nature. The qualities of an error-detecting code are often judged according to its detecting capacity in this very class of errors, assuming of course that the single errors are detected anyhow. Mathematically, it turned out that the decimal codes were especially difficult in this respect. Some authors thought that they proved that decimal codes detecting all single errors as well as all transpositions, were non-existent. (see 46, 39). These "proofs" came fortunately after that the present author had constructed such a code. (see 51).

There are several minor classes of double errors, which are important since codes detecting all single errors and all transpositions may be completely immune for these classes. Their frequency is small, say 0.5 to 1.5% of all errors.

These classes are:

2.2) The twin errors. These are adjacent errors of the type aa→bb. They can easily be explained, for in case one "a" is misread as a "b", the other one is likely to be misread too. Also if one is punching blindly, it is logical that, if a finger is on the wrong key for the first "a", that then the second one will be treated or rather mistreated in the

same way.

2.3) The jump transpositions. A jump transposition is the interchange of 2 digits, jumping over a third one, like abc→cba. It could also be called a reversion, since the order of the 3 digits gets reversed. Its psychological explanation can perhaps be sought in an auditive echo.

2.4) The phonetic errors. The above mentioned clearing-house error samples reveal that in the adjacent double errors, the errors of the type ab →ca or vice versa, occur much more than chance predicts. Among these the errors with b=0 and c=1 are again much more numerous than expected. These errors are called phonetic. They might be explained by the phonetic resemblance when the pairs a0 and 1a are pronounced. This is of course dependent on the language, but it holds in English, Dutch and German. This explanation is strenthened by the fact that the errors 12 →20 and vice versa are indeed much less frequent. It would be interesting to know how this is in the French speaking countries. It is also an open question whether an oral communication link is needed or whether punch typists with an auditive memory can be responsible for this error type.

2.5) The jump twin errors. These errors are of the form aba→cbc. Their frequency is, as is to be expected, lower than that of the twin errors, say 50%. Their explanation may be the same. The frequency of more remote twin errors, like a..a → c..c is very much lower. This is also the case with the interchange of digits over more than 1 digit.

3) The third class consists of omitting or adding a digit to the code word. The frequency lies somewhere between the 10 and 20%. The vast majority consists of the omission of one digit, where the last position again seems to be the most vulnerable one. It is also striking that the 0 is the decimal which is most easily dropped. However, since there seems to be a tendency to allocate "beautiful" numbers, ending with one or more zero's, to important customers, like tax collectors, there definitely is a bias if the statistics are drawn from banking and clearing systems. It is also remarkable and equally dubious, that the forgotten digits often were members of a sequence of identical ones. It is an illustration of the ancient theorem that beauty is dangerous.

4) The random errors. The fourth catagory of errors is called random, since it consists of those errors for which there is apparently no relation between the correct number and the erroneous one. It is also

believed that all numbers are equally susceptible for this error type. The random errors are both pleasant and nasty. They are pleasant, because all redundancy helps to detect them. They are nasty, since it is impossible to design a code which would do any better than any other code. There is also a very important assumption made, hidden in the word "apparently". For there may very well be no relation between the code numbers as such, but there can be a hidden semantic relation. E.g., both numbers can belong to the same person, one being his account number and the other his telephone number. It also can happen that the two numbers are adjacent in some code book. It is difficult to trace that kind of errors down without employing a full time detective. All this would not be so serious as long as these "semantic" errors behave like random errors, but there is every reason to believe that this sort of error will prove to be immune for all detection systems. If somebody is copying the wrong number correctly, he will do so too if the number is one of an error detecting code. These immune errors may turn out to be one of the criteria for how far one has to go in the imporvement of the detection capacity of a code. Suppose that a certain system has to cope with 100 errors a day, 50 of which being immune. Now one might be interested to cut this down to 55 undetected errors, the same 50 immune ones included at the cost of one more check digit. However to cut this down to 50.5 at double cost might be unattractive. The immune errors form some kind of basic noise level.

The total frequency of the random errors varies considerably, depending on the nature of the system. if the code numbers are more or less used publicly this class might be much bigger than for those systems where the numbers are more privately used, like passport numbers etc.. For the clearing house systems 5 to 15% has been measured. The percentage of the immune errors, though more important, is unknown.

5) Finally, there remains the traditional class called miscellaneous. It contains collector items like; aba → bab; abcd → cdab; aaaa → bbbb. All are rare and mostly difficult to detect for 100%. Occasionally some defy detection, so that in studies of the undetected errors of a certain code, these rare errors might seem significant.

0.6. A sample of errors.

The main sample of errors, available to the author, concerns errors made in a non-fixed length code. The total size of the sample is 22733 pairs (consisting of a good code word and an erroneous one). The sample is divided according to the length of both numbers, as follows:

Both	numbers	7	digits	8
"	"	6	"	12112
"	"	5	"	3333
"	"	4	"	1774
"	"	3	"	139
"	"	2.	"	25
Unequal length				5342

There were 2343 cases of one forgotten digit.
The analysis of the largest group will be given here as an illustration. The distribution according to the number of places on which the words of each pair differ is:

1 place	9574	or	78.9%
2 places	1870	or	15.6%
3 places	169	or	1.4%
4 places	118	or	1.0%
3 places	219	or	1.8%
4 places	162	or	1.3%
	12112		

A further analysis of the single errors reveals that the rightmost digit is affected most frequently. The distribution according to the position of the error, counting from the right, is:

position 1 :	2854
position 2 :	2296
position 3 :	1270
position 4 :	929
position 5 :	1503
position 6 :	722
	9574

The following matrix gives the transition frequencies of the ten decimals. The 125 in row 4 and column 6 means that 125 times a "4" became a "6".

	0	1	2	3	4	5	6	7	8	9	
0	0	127	27	42	19	47	286	4	234	329	1115
1	163	0	124	30	105	47	54	70	11	15	619
2	56	127	0	380	61	74	32	101	24	32	887
3	95	50	340	0	49	357	53	48	181	63	1236
4	60	161	132	64	0	134	125	175	33	189	1073
5	57	62	84	431	111	0	185	19	91	30	1070
6	210	77	39	51	104	169	0	61	180	43	934
7	13	95	93	27	164	46	55	0	20	167	680
8	210	29	68	221	44	119	167	20	0	73	951
9	292	21	53	62	154	82	61	184	100	0	1009
	1156	749	960	1308	811	1075	1018	682	874	941	9574

It would be highly interesting to know which properties, of this matrix, are independent of the system from which the errors are drawn.
The restricted single errors total 2923, which is higher than the expected 2/9-th of 9574. The digit "3" seems to be the black sheep of the decimals.
The double errors have also been subjected to a further analysis. From a technical (and probably also from a psychological) point of view it is interesting to know whether the double errors tend to come in bursts. The following distributions according to the distance of the errors in the words, has been found.

Distance	1	(adjacent positions)	1595
"	2	(x.x, jump errors)	177
"	3	(x..x)	71
"	4	(x...x)	18
"	5	(x....x)	9
			1870

This statistic strongly suggests that the errors are dependent. The 1595 burst errors are subdivided into:

Transpositions	1237
Twin errors	67
Phonetic errors	59
Rest	232
	1595.

The 177 jump errors are divided into:

Jump transpositions	99
Jump twin errors	35
Rest	43
	177.

The distribution of the phonetic errors according to their position in the code word is as follows: $\underbrace{x.x}_{15}\overset{0}{.}\underbrace{x.x}_{9}\overset{1}{.}\underbrace{x.x}_{34}$.

The absence of the phonetic errors on the odd positions may be explained by the habit of quoting the words in pairs of decimals. The distribution of the errors $1x \to x0$ and $x0 \to 1x$,

over x is:

x	2	3	4	5	6	7	8	9
#	3	12	10	9	4	7	1	13

It is typical that 8 has such a low frequency, because in the Dutch language 80 is "tachtig" but 18 is "achttien" in contradistinction with the English and German which are consistent with "eighty" and "eightteen" and "achtzig" and "achtzehn" respectively.
The multiple errors are mostly errors of the random type and as such they defy analysis.

0.7. Detection versus prevention.

No matter how good the error detecting capacity of a check system is, one will still be interested in minimizing the number of errors. The available measures, which belong mainly to the realm of human engineering fall outside the scope of this monograph. There is however also a mathematical approach to the problem of error prevention. This approach is based on the non-uniformity of the distribution of the errors over the code words. By selecting a code C in such a way that the overall error chance is minimal, a certain error prevention is achieved. The more

error prone code words are excluded. In the Dutch population register system those code words, having equal decimal digits on adjacent positions, are avoided, since these code words are considered to be more error prone than the other ones.
The available statistics are however still insufficient to tackle this problem effectively.
Another virtually unknown factor is the increase of the error chance because of the added check digit. It is obvious that a high redundancy may very well successfully lower the percentage of the undetected errors, but it will also lower the percentage of correct code words. The detection becomes, if the redundancy increases, in a certain sense, less effective. The reason is, that how longer the code word is, the less information the detection of an error provides. So will it be a small surprise to learn that a certain book contains an error.
The ultimate goal of detection is of course a correction. This can often only be done by feedback towards the source of the error. In systems with a decentralized input and a parallel processing, it is a customary procedure to reject the erroneous inputs, so that the rest can be processed. If this rest is not the bulk of the workload, or if the system is processing serially, it becomes desirable to have an on-line correction. The problem arises to construct codes with the property that such a correction, which can never be infallible, is at least most likely.

0.8. Error correcting codes.

An error correcting code is a redundant code C, along with a decision scheme which associates with certain inproper code words a proper one, which is called the corrected code word. This association can in principle be done quite arbitrarily, but it is natural to do it in such a way that each code word is imbedded in a set of words which can be obtained by making an error, of a certain type, in said code word. If the code is such that these sets are mutually disjoint, then an error of that type can be corrected by the convention that if a word of such a set is received then the only proper code word of that set is taken as the corrected word. One could also say that in such a case the coding is not unique, since to each object a whole set of code words is

allocated. The code, though no longer unique, has still to be unambiguous and this is so as soon as the associated sets are mutually disjoint. Hence to each $x \in C$ there belongs a set A(x) , with the property that from $x \neq y$ it follows that $A(x) \cap A(y) = 0$. Let V be the union of all A(x), thus $V = \bigcup_{x \in C} A(x)$. If V=U then the code is said to be perfect or close packed. In V there is an equivalence defined by the classes A (x) and each word of V is equivalent with just one word of C. This defined a mapping Φ of V on C. The correction procedure corrects each word w of V into $\Phi(w)$. If a word outside V is received then the error is detected, but cannot be corrected. This cannot occur if V=U, i.e. if the code is close packed. The term perfect is less appropriate, since it is in a way not the code which is perfect but the correcting scheme because it corrects every error. This property may be desirable for the applications in the serial processes, but not for the systems with parallel processing where the correction is only needed to secure that the bulk of the input can be processed. In order to appreciate this point it should be noted that an error correcting code only guarantees the correction of a certain type of errors. In real life however also errors of other types are bound to occur. A perfect correcting scheme will "correct" these errors by introducing an error of the protected type. It may therefore be a good policy to choose V deliberately so small that certain errors will never be "corrected".
The code of the Dutch population register system is a single error correcting code which does not "correct" the transpositions.
The type of random errors is the stumbling block, since a random error is never guaranteed to fall outside V. In fact a random error correcting code C has only one code word, since A(x)=U for all x. This trivial code is always perfect.

0.9. Disjoint codes.

Let C again be a redundant code in a space U, It is often possible to find one or more codes C' with the same detecting capabilities as C, but disjoint with C. As will be seen later on, decimal codes defined by a check equation will split up the space U into 10 mutually disjunct codes (, or in general k,if one is working modulo k). In section 0.7 it was pointed out that, though these codes are equivalent detection-

wise, they may be different as to the overall error chance. There is another way in which these disjoint codes may be useful for the applications. Suppose that two operating systems, perhaps sharing many customers, need an error protection for their codes. By adding a check digit to the existing code words much of the cost of recoding can be avoided. If these systems draw the check digit from disjoint codes they have the additional advantage that each valid number of one system is invalid for the other system. This might eliminate a source of seemingly random errors.

Another application might be a group of branch offices of a large bank with a central administration. If disjoint codes are used for the clients of the various branch offices, then one would have a protected code without using more digits. The traditional solution would use the first digit to designate the branch office without giving any error detection possibility within the local administration. It is an elegant way of setting the redundancy at work. In cases with more than 10 subsystems a higher modulus check might be useful (see section 2.3).

0.10. Better error detection by random use of a code.

In section 0.3 it was argued that the natural redundancy, which is usually present since the codes are seldom used to full capacity, may lead to error detection during the processing. It would be an advantage if this detection could be done during the input stage. This can be achieved by a controled use of the code. It is not uncommon to use only the first interval, under lexicographical ordering, of the code. Suppose that some system with 600000 customers uses a code with 7-digit decimal code words. Using only the first 600000 numbers guarantees that an error which yields a higher number is detected at the input if the proper measures are taken. The protection procured in this way is however primarily aimed at the first (least vulnerable) decimal. Much better in this respect is the pseudo random use of the code, which can be accomplished in the following manner. With the aid of a reversible

deciphering one can shuffle the code words and by using the first (in example above, 600000) numbers, the used part of the code is (pseudo) randomly distributed over the code. Now a code word received at the systems input can be reshuffled and if it does not belong to the first 600000 an error is detected. In the code of the Dutch population register system this feature is incorporated. The reversable deciphering is done with a feedback shiftregister, working in the field of the complex ternary numbers, i.e. the complex numbers of the form a+bi with $a,b \in \{0,1,2\}$.

CHAPTER 1. Bounds for redundant codes.

1.0 Definitions.

In this chapter some concepts are introduced to facilitate the discussion of redundant codes.

An error type is essentially a mapping of a set of (potential) code words into the class of its subsets. To each $a \in U$ there corresponds a set $E(a) \subseteq U$ of all those words which can be derived from a by an error of the given type. The set E(a) may be empty. In the case of the random errors each $a \in U$ is mapped on the set U-a. A code C is called E-proof if $E(a) \cap C = 0$ for all $a \in C$. An E-proof code C admitts a correcting scheme for the error-type E if $E(a) \cap E(b) = 0$ for all $a, b \in C$, with $a \neq b$. It is then called an error correcting code. If moreover $\bigcup_{a \in C} E(a) = U - C$ then the correcting code is called perfect or close-packed. An E-proof code C is called maximal if there does not exist an E-proof code C' which properly contains C. If such is the case it follows that $E(b) \cap C \neq 0$ for each $b \notin C$. Schauffler (43) calls such a code closed (abgeschlossen) with respect to E.
An E-proof code C is called optimal if there does not exist an E-proof code in U, with more words than C. An optimal code is necessarily maximal. A code C is said to be p% E-proof if

$$p/100 = \sum_{a \in C} |\bar{C} \cap E(a)| \Big/ \sum_{a \in C} |E(a)| .$$

The redundancy of a code C in an m-ary space U is defined as $\lg_m(|U|/|C|)$ digits or $\lg_2(|U|/|C|)$ bits.
An error-type E is called symmetric if from $a \in E(b)$ it follows that $b \in E(a)$. Most of the error-types mentioned in the introduction are symmetric. The type of the forgotten digits is an exception.
For symmetric error-types a metric can be defined. The distance between a and b is the minimal number of errors (of the given type) which have to be made in a, in order to get b. It is called the E-distance and denoted by $d_E(a,b)$. The subscript E will often be dropped. More formally: $d(a,a)=0$ and $d(a,b)=k$ if there exists a chain $a_0, a_1, \dots, a_k$ with $a_0=a$ and $a_k=b$ such that $a_{i+1} \in E(a_i)$; $k > i \geq 0$ and if there does not exist a shorter chain with that property. If no chain exists at all the

distance is per definition infinite.
The E-distance is properly called a distance since:
$d(a,b) \geq 0$ and

i) The reflexive law $d_E(a,a)=0$;

ii) the symmetric law $d_E(a,b)=d_E(b,a)$ and

iii) the triangle inequality $d_E(a,b)+d_E(b,c) \geq d_E(a,c)$

are fulfilled.

i) is obvious, ii) follows directly from the symmetry of the error-type E and iii) follows from the fact that the concatenation of the chains from a to b and from b to c forms a , not necessarily minimal chain from a to c. The definition of distance is a straightforward generalization of the Hamming distance for the single bit errors in binary codes.
If all distances are finite the space U is called connected with respect to E. Otherwise U falls apart into connected components. The diameter of a connected space U is $\max_{a,b \in U} d_E(a,b)$. For the random errors the diameter of every space is 1. For the single errors the diameter is equal to the dimension of U.
The greatest possible diameter is $|U| - 1$ since that is the length of the longest chain in U. The following examples show that this diameter is possible. Suppose that the code words of U are listed somehow in a codebook. Let the type of error be that of taking the list item directly preceding or following the correct one (restricted look-up errors). Another example is that U consists of a set of consecutive integers with respect to the errors of one unit in the arithmetical sense. The E-distance of two different words a and b, of an E-proof code C, is at least 2. If it were less, then $b \in E(a)$, but for an E-proof code $E(a) \cap C=0$ holds. If C admitts a correcting scheme the E-distance between any two words is at least 3, for otherwise there would exist a word c such that $d(a,c)=d(c,b)$ and hence $c \in E(a) \cap E(b)$.
A code C is said to have a minimum distance k when $\min_{a,b \in C, a \neq b} d_E(a,b)=k$.

In view of the definition of distance it will be clear that a code

with minimum distance 2e+1 will admitt a correcting scheme for e errors of the type E. It also will detect 2e or less errors.

An E-ball of radius k and center a is the set of all words x satisfying $d_E(a,x) \leq k$. It is denoted by $S_E(a,k)$ The difference between a ball with radius 1 and the error sets E(a) is clearly that the latter does not contain a; hence $S_E(a,1)=E(a) \cup \{a\}$.

Let $E^i(a)$ be the set of those words obtainable from a by making i mistakes of the type E, but which cannot be obtained by making fewer than i errors. Thus $E^0(a)=a$ and $E^1(a)=E(a)$ and $E^i(a)=S_E(a,i)-S_E(a,i-1)$, for $i>0$.

Conversely, $S_E(a,e)= \bigcup_{i=0}^{e} E^i(a)$ for $e \geq 0$. From the definition it follows immediately that $E^i(a) \cap E^j(a)=0$ for $i \neq j$ and therefore $|S_E(a,e)| = \sum_{i=0}^{e} |E^i(a)|$.

An error-type is called uniform if $|E(a)| = |E(b)|$ for all $a,b \in U$. Single errors are of the uniform type, whereas the transpositions are non-uniform ($E(13)=\{31\}$ and $E(22)=0$). An error-type is strongly uniform if $|E^i(a)| = |E^i(b)|$ for all $i \geq 0$ and all $a,b \in U$.

1.1 <u>Some upper bounds for minimum distance codes</u>.

From the definition of an error correcting code it follows that $S(a,1) \cap S(b,1)=0$ for all $a,b \in C$ with $a \neq b$. An immediate consequence is the relation:

$$| U | \geq \sum_{a \in C} | S(a,1) | .$$

Therefore:

theorem 1.1.0 The redundancy of a minimum distance 3 code is at least $lg_m (\sum_{a \in C} | S(a,1) | / |C|)$ digits.

This bound is a generalization of the sphere packing bound, as it is known in the literature on the binary codes with respect to the single bit errors. For uniform error-types the bound is simplified into $lg_m |S(a,1)|$.

The obvious generalization is the

theorem 1.1.1 The redundancy of a minimum distance 2e+1 code is at least $lg_m(\sum_{a \in C} |S(a,e)|/|C|)$ digits.

Proof: Let a and b be two words of a minimum distance 2e+1 code C, then $S(a,e) \cap S(b,e)=0$, for otherwise there would exist a word $c \in U$ with the

property $d(a,c) \leq e$ and $d(b,c) \leq e$. From the triangle inequality it then would follow that $d(a,b) \leq 2e$, which contradicts the minimum distance property of C. The disjunctness of the spheres and the relation $\bigcup_{a \in C} S(a,e) \subseteq U$ give $\sum_{a \in C} |S(a,e)| \leq |U|$. After division by $|C|$ and by taking the m-logarithm of both sides of this relation the theorem is found. These theorems are especially helpful for proving the nonexistence of certain codes.

The Hamming codes are examples of perfect minimum distance 3 binary codes. These codes enable the correction of single errors. Perfectness of codes is a mathematical nicety, which has from a practical point of view the disadvantage that all other errors are "corrected" by introducing another error. The point is of course that the non-perfect codes have a higher redundancy. Perfect binary codes for correcting more than 1 single error are collector items. (45).

Finding an optimal error correcting code is a matter of packing as many balls $S(a,e)$ as possible in the space U. For a strongly uniform error-type a close-packed code is necessarily optimal. For a non-uniform error-type it is conceivable that a perfect code is not optimal since the latter might have many small balls whereas the perfect one possibly covers U with a few large balls. For the even minimum distance codes it is not simply a matter of packing balls since these may now overlap each other. The question is how this overlapping can be done effectively.

Consider two points a and b of a minimum distance 2e code C in the space U. Suppose that $d(a,b)=2e$, then $S(a,e-1) \cap S(b,e-1)=0$ and $E^e(a) \cap E^e(b) \neq 0$. The space U is split up into 3 types of points i.e.

i) The points of the balls with center in C and radius e-1,

ii) The points contained in the sets $E^e(a)$ with $a \in C$,

iii)The other points.

Denote these mutually exclusive sets by U_1, U_2 and U_3 respectively.

Then $U_1 = \bigcup_{a \in C} S(a,e,-1)$ and $U_2 = \bigcup_{a \in C} E^e(a)$.

The following relations hold:

$$|U_1| = \sum_{a \in C} |S(a,e-1)|, \quad |U_2| \leq \sum_{a \in C} |E^e(a)| \text{ and } |U_3| \geq 0.$$

Now define $c(a,e)$ as the maximal number of points a_i such that

$d(a,a_i)=e$ and $d(a_i,a_j)\geq 2e$ for $c(a,e)>i>j\geq o$. It will be called the covering index of a. Since each $x\in U_2$ can be in at most $c(x,e)$ sets $E^e(a)$ the relation $\sum_{x\in U_2} c(x,e)\geq \sum_{a\in C} |E^e(a)|$ holds. Let $c_e = \max_{x\in U} c(x,e)$, then $|U_2|\cdot c_e \geq \sum_{x\in U_2} c(x,e)\geq \sum_{a\in C} |E^e(a)|$.

Consequently $|U-U_1|\, c_e=|U_2+U_3|\cdot c_e\geq |U_2|\cdot c_e\geq \sum_{a\in C} |E^e(a)|$. Combining this relation with $|U-U_1|=|U|-\sum_{a\in C}|S(a,e-1)|$ gives

$|U|/|C|\geq \sum_{a\in C} \{|S(a,e-1)| +|E^e(a)|/c_e\} /|C|$.

In this way a lower bound for the redundancy has been found. For strongly uniform error-types this bound is simplified into $|S(a,e-1)|+|E^e(a)|/c_e$ where a is chosen arbitrarily in C.

The result is formulated as:

theorem 1.1.2 The redundancy of a minimum distance 2e code is at least $lg_m\{ \sum_{a\in C}\{|S(a,e-1)| +|E^e(a)|/c_e \}/|C|\}$ digits.

For strongly uniform error-types this bound is simplified into $lg_m(|S(a,e-1)| + |E^e(a)|/c_e)$, with $a\in C$.

1.2 Single errors.

Let the type of the single errors be denoted by E_1. Let n be the dimension of the space U of m-ary words. The set $E_1(a)$ consists of all words which differ from a on only one position. There are m-1 possibilities per position and thus $|E_1(a)|=n(m-1)$ and $|E_1^i(a)|=\binom{n}{i}(m-1)^i$. Two words a and b of an E_1-proof code differ therefore on at least two places and that is why such a code is sometimes called bidifferent.

Theorem 1.2.0 The redundancy of a bidifferent code in an m-ary space U is at least 1 digit.

Proof: Suppose $lg_m(|U|/|C|)<1$, then $|C|>|U|/m=m^{n-1}$, where n is the dimension of U. Since there are m^{n-1} different words with n-1 positions, it follows that C contains at least two words say a and b, which are identical on the first n-1 positions. Hence $b\in E_1(a)$ and consequently C is not E_1-proof.

Theorem 1.2.1 There do exist bidifferent m-ary codes with a redundancy of 1 digit.

Proof: Let U be the space of all m-ary words with n positions. One may assume that the symbols of the words stand for the residue classes modulo m. If this were not the case one can first make a 1-1-correspondence between the symbols and these residue classes. Now let $a_1a_2...a_n$ be a word of U and consider the sum $s=\sum_{i=1}^{n} a_i$. There are m values possible for s and thus the words of U are divided into m classes according to that value. These classes all have the same number of elements. This is so since the m different words which are equal on say the first n-1 positions clearly are in different classes. From this it follows that each of these classes is a code with 1 digit redundancy. Moreover since words differing from each other on only one place cannot have the same digit sum modulo m, each one of these codes is bidifferent. In view of the preceding theorem they are also optimal.

Just for curiosities sake two examples of maximal bidifferent codes with a higher redundancy will be given.

000, 101, 202, 303, 404, 555, 656, 757, 858, 959,
011, 112, 213, 314, 410, 566, 667, 768, 869, 965,
022, 123, 224, 320, 421, 577, 678, 779, 875, 976,
033, 134, 230, 331, 432, 588, 689, 785, 886, 987,
044, 140, 241, 342, 443, 599, 695, 796, 897, 998;

This is 50 word 3-digit decimal maximal bidifferent code. A similar one with 52 words is given in the next example.

000, 101, 202, 303, 404, 505, 666, 767, 868, 969,
011, 112, 213, 314, 415, 510, 677, 778, 879, 976,
022, 123, 224, 325, 420, 521, 688, 789, 886, 987,
033, 134, 235, 330, 431, 532, 699, 796, 897, 998,
044, 145, 240, 341, 442, 543,
055, 150, 251, 352, 453, 554;

The construction of an optimal bidifferent code is equivalent with a generalization of the problem of the rooks, well-known from recreational mathematics. (see 27, p. 240). It is the problem of how to place m rooks on an m-th order chessboard so that no rook can capture any other one in a single move. The generalization uses a n-dimensional board with generalized rooks, say hyperrooks. The equivalence is obvious since the set U

of all m-ary words can be taken as an n-dimensional m-th order chessboard. A hyperrook placed on a field a covers exactly the fields of the set $E_1(a)$. A bidifferent code is therefore a set of fields where hyperrooks can be placed such that they cannot take each other in one move. The code is maximal if there is no uncovered field left in U. The optimal codes, having m^{n-1} words are the solutions of the rook problem. For n=2 all maximal solutions are optimal, but the examples mentioned above show that such is no longer the case for n >2. Other error-types correspond in this terminology with fancy chessman, having esoteric ways of moving. Theorem 1.2.0 can also be derived from theorem 1.1.2. The covering index c(a,1) is obviously n for every a, since this is the maximal number of points differing from a on one place and from each other on two places. Thus as $|E_1(a)| = n(m-1)$ holds it follows that the minimum redundancy is $\lg_m(1+n(m-1)/n)=1$ digits.

Theorem 1.2.2 The redundancy of a n-digit minimum distance 2e+1 m-ary code is at least $\lg_m(\sum_{i=0}^{e} \binom{n}{i}(m-1)^i)$ digits.

The proof follows at once from theorem 1.1.1 by substituting $\binom{n}{i}(m-1)^i$ for $|E^i(a)|$.

This bound is known in the binary case from Hamming (17). Let the size of an n-digit m-ary code with minimum distance d with respect to the single errors be denoted by A(m,n,d).

Hamming proves in the same paper that A(2,n,2e)=A(2,n-1,2e-1). His reasoning is simple: Suppose a minimum distance 2e code with n bits is given. By chopping off one bit a n-1 bit code is formed. Obviously this code has at least a minimum distance 2e-1 since the chopped-off bit contributed at most one unit to the distance. Thus A(2,n-1,2e-1) $\geq$ A(2,n,2e). Conversely when a minimum distance 2e-1 code with n-1 bit is given, a n-th bit can be added such that the number of ones in each code word becomes even (parity check). The words which were at a distance 2e-1 from each other are now necessarily different on the n-th position, as 2e-1 is odd. For the pairs which had already a greater distance the n-th bit is irrelevant, so that a minimum distance 2e code with n bits is derived. Hence A(2,n,2e) $\geq$ A(2,n-1,2e-1) and therefore A(2,n,2e)=A(2,n-1,2e-1). Only the first part of this reasoning is valid for the higher number bases and thus:

Theorem 1.2.3 $A(m,n-1,2e-1) \geq A(m,n,2e)$ for $m \geq 2$.

A counter example will show that the converse of the theorem above is not true. Consider a 4-digit minimum distance 3 ternary code. Substitution in 1.2.2 gives $A(3,4,3) \leq 3^4/(1+4(3-1))=9$. The 9 word code: 0000, 0111, 0222, 1021, 1102, 1210, 2012, 2120, 2201 is therefore optimal.

This code is in fact a Graeco-Latin square of order 3.

	0	1	2
0	00	11	22
1	21	02	10
2	12	20	01

A 5-digit minimum distance 4 ternary code would, if it had 9 words, be the same as 3 Latin squares of order 3, such that each pair forms a Graeco-Latin square of the same order. But it is well-known that this is impossible (see Ryser 42 p.80).

With the aid of theorem 1.1.2 an upper bound for $A(m,n,2e)$ can be derived which is better than the combination of the theorems 1.2.2 and 1.2.3, if $e \nmid n$ and not worse if $e|n$.

Theorem 1.2.4 $A(m,n,2e) \leq m^n/(\sum_{i=0}^{e-1} \binom{n}{i}(m-1)^i + \binom{n}{e}(m-1)^e/\text{entier}(n/e))$.

Proof: The maximal number of words differing from a certain word a on e places and from each other on at least 2e places is clearly equal to entier(n/e) . The theorem now follows by substitution of $E_1^i(a)$ and c_e.

That this bound is an improvement can be seen by comparison.

$$\sum_{i=0}^{e-1} \binom{n}{i}(m-1)^i + \binom{n}{e}(m-1)^e/\text{entier}(n/e) \geq m. \sum_{i=0}^{e-1} \binom{n-1}{i}(m-1)^i$$

$$\text{or } \binom{n}{e}(m-1)^e/\text{entier}(n/e) \geq \sum_{i=0}^{e-1} \left(m\binom{n-1}{i} - \binom{n}{i}\right)(m-1)^i =$$

$$\sum_{i=0}^{e-1} \binom{n-1}{i}(m-1)^{i+1} - \sum_{i=0}^{e-1} \binom{n-1}{i-1}(m-1)^i = \binom{n-1}{e-1}(m-1)^e.$$

Finally $\binom{n}{e}/\binom{n-1}{e-1} = n/e \geq$ entier(n/e) which is obviously true and the equality sign therefore only holds if $e|n$.

In connection with $A(2,n,2e)=A(2,n-1,2e-1)$ an interesting corollary follows. For, $A(2,n,2e+1)=A(2,n+1,2e+2)\leq 2^n/\sum_{i=0}^{e}\binom{n}{i}$ where the equality sign only holds if $e+1\,|\,n+1$, hence

Corollary :A close-packed minimum distance $2e+1$ binary code is only possible if $e+1\,|\,n+1$.

This corollary easily shows the non-existence of a 90 bit minimum distance 5 code. Though $1+\binom{90}{1}+\binom{90}{2}=2^{12}$ and $\binom{5}{2}\,|\,\binom{90}{3}$ are both valid (the first condition is Hamming's sphere packing condition whereas the second comes from theorem 1 of Shapiro and Slotnick (45)), $2+1\,|\,90+1$ does not hold.

The upper bound for $A(3,5,4)$ becomes $3^5/(1+5\text{x}2+10\text{x}2^2/2)=243/31\;\;7.9$.

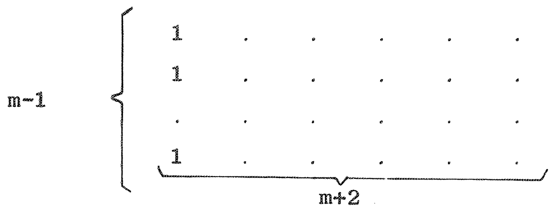
The true value for $A(3,5,4)$ is 6 as is shown by the example: 00000, 01111, 11202, 12120, 20221, 22012. That this code is optimal can be seen as follows: Suppose that 3 words had the same symbol on the same position, say a 0 on the first position. Then these words have to differ on all other positions. Since permutations of the symbols per position do not change the distance between the words, those 3 words may be taken as 00000, 01111, 02222. These words form however a maximal code, for each 5-digit word has to have at least 2 equal symbols on the last 4 places and therefore cannot differ on 4 places with those 3 words. Consequently each symbol can occur at most twice on each position, which is so in the example. Hence 6 is the maximal number of code words. The same reasoning shows:

Theorem 1.2.5 $m(m-1)\geq A(m,m+2,m+1)$.

For $m=4$ this gives $12\geq A(4,6,5)$, but this bound can be sharpened by remarking that, for $m>3$ an optimal code cannot have $2(m-1)$ words which share on a certain position $m-1$ symbols of one kind, say a 0 and $m-1$ symbols of another kind, say a 1. As the first $m-1$ words one may again take

$$m-1\left\{\begin{array}{cccccc} 0 & 0 & . & . & . & 0 \\ 0 & 1 & . & . & . & 1 \\ . & . & . & . & . & . \\ 0 & m-2 & . & . & . & m-2 \end{array}\right.$$

and as the second $m-1$ words one may take:



In each of the words, starting with a "1", the m-th symbol (i.e."m-1") has to occur at least twice, as there are m+1 places left and as the symbols from 0 to m-2 may occur only once in each of those words. But the m-th symbol itself may occur only once on each position in the words having already a "1" in common. Thus there are 2(m-1) positions required and $2(m-1) > m+1$ for $m > 3$. Thus:

theorem 1.2.6 $(m-1)+(m-1)(m-2)=(m-1)^2 \geq A(m,m+2,m+1)$ for $m > 3$. That this bound is sharp, at least for m=4, is shown by the example:

0	0	0	0	0	0
0	1	1	1	1	1
0	2	2	2	2	2
1	3	3	2	1	0
1	2	0	3	3	1
2	3	0	1	2	3
2	1	3	3	0	2
3	3	1	0	3	2
3	0	2	3	1	3

Thus A(4,6,5)=9.

For m >4 a better upper bound is given in:

Theorem 1.2.7 If $d \geq n(m-1)/m$ then $md/(md-n(m-1)) \geq A(m,n,d)$.

Proof: Let an n-digit m-ary code with minimum distance d have k words. Any pair of words has on at most n-d places the same digits. Call an occurrence of equal digits on a same position a match. Since the number of word pairs is $\binom{k}{2}$ there are at most $(n-d)\binom{k}{2}$ matches in the code. Now let k_{ij} be the number of code words which have the i-th digit on the j-th position. The number of matches on the j-th position is then $\sum_{i=1}^{m} \binom{k_{ij}}{2}$. Now the minimum of this sum is reached if all k_{ij} are equal, for it is well-known that $\min(\sum_{i=1}^{m} x_i^2)$ with $x_i \geq 0$

and $\sum_{i=1}^{m} x_i = k$ is reached for $x_i = k/m$. Thus

$$\sum_{i=1}^{m} \binom{k_{ij}}{2} = \sum_{i=1}^{m} (k_{ij}^2 - k_{ij})/2 \geq m(k^2/m^2)/2 - k/2 = (k^2/m-k)/2.$$

So that the total number of matches is at least $n(k^2/m-k)/2$ and thus

$(n-d)\binom{k}{2} \geq n(k^2/m-k)/2$ has to hold. Division by k/2 gives

$(n-d)(k-1) \geq n(k/m-1) = n(k-m)/m$. After shuffling terms $md \geq k(md-(m-1)n)$

the theorem is proved.

Corollary: $m(m+1)/2 \geq A(m,m+2,m+1)$. For $m \geq 5$ this bound is better than the one of theorem 1.2.6. It is not known to the author whether $A(5,7,6)=15$ is true.

In the binary case the bound of theorem 1.2.7 is known as the Plotkin bound (see 38). This bound is also known to be true if m is a prime power (see 37).

1.3. Check digits.

Adding a check digit to the code words is perhaps the best known method for introducing redundancy. A check digit is a digit which is determined by the other digits. The latter are free to take any value and are for that reason called information digits. Let M(m,n) be the set of all m-ary code words with n digits. If a code word of M(m,n) is extended by a check digit then it becomes a member of M(m,n+1). Thus M(m,n+1) contains a subset C of such extended code words and $|C|=|M(m,n)| = m^n = |M(m,n+1)|/m$. The redundancy of C is 1 digit. The introduction of a check digit is an erderly way to define a subset with a redundary of 1 digit. If the check digit can be expressed as a function which admitts a simple computation, it may also be a concise way of doing it. Moreover it is often possible to derive the detecting properties from the properties of the function. It is in general not true that every code with 1 digit redundancy can be considered as a code with a check digit. Examples are the 3 bit binary code {000,001,101,111} and the 2 digit ternary codes {10,21,20 } or {00,01,11} . That for instance the third bit in the first code cannot be a check bit follows from the observation that in the first two words the first two bits are equal and therefore cannot give different check bits. For bidifferent codes however the converse is in fact true.

Theorem 1.3.0 In a bidifferent code with 1 digit redundancy each digit can be considered as check digit.

Proof: The i-th digit can be considered as a check digit if all words are different on the other n-1 positions. This is obviously so because of the bidifference of the code.

There are various ways to define a check digit or what amounts to the same, to define a function on a finite set. The most general one is the method of the table look-up. The arguments of the function are simply listed and the proper function-value is entered behind each argument. The method though general is not attractive for applications, the very simple ones excluded. It is in cases of any interest virtually impossible to construct a code with precribed properties. Moreover it is only possible by means of a large memory to have automatic detection of errors. It is therefore natural to apply check digits defined by some sort of a formula, or an algorithm. A simple example is the parity check in the binary case. The check bit is chosen in such a way that the number of 1's in the code words (check bit included) is even. The parity check is well-known and finds wide application in the computer design. Binary codes however are not popular for use by human beings. It is mentioned here only as an illustration. It may be interesting to note that the complement of the parity check is a disjoint code, called the imparity check. The space U of the n-bit code words is divided into two equal parts, i.e. the words with an even number of 1's (the parity check code) and the words with an odd number of 1's (the imparity check code). The codes are essentially the same since the inversion of one bit (i.e. interchanging 1 and 0) makes the codes identical. The parity check can easily be generalized for an arbitrary number base m. Let a_i be the symbol on the i-th position of an (n-1)-digit m-ary code word. An n-th digit can then be found such that $\sum_{i=0}^{n} a_i = 0 \pmod{m}$ or $a_n = -\sum_{i=0}^{n-1} a_i$. This check is called the straight modulo m check. It was used in the proof of theorem 1.2.1. For m=2 it is the parity check. The detecting properties of this check will be discussed in ch.2. At present it only serves as an example for the generation of a check digit by means of a formula. In that light it is important to note that the check digit can be found recursively as follows: Take $c_0=0$ and $c_i=c_{i-1}-a_i$ for $i>0$, then c_{n-1} is the check

digit. If the digits of a word are fed into a cyclic m-counter one after the other, the counter will end in the initial state after that the check digit has been entered. The state of the circuits is in each stage giving the value of the check digit no matter how many digits are fed into it. This is a very desirable property for the technical implementation of check digit verifiers. The example of the cyclic m-counter is simple in the sense that its reaction is independent of the positions of the various digits. The drawback is of course that the code is of no high quality. Later on codes will be introduced with crooked and position dependent "m-counters". Under a crooked m-counter is understood a (sequential) circuit with m states s_i and m possible inputs a_i; $0 \leq i \leq m-1$ such that two conditions are fulfilled i.e.: i) Any input acting upon the circuit in different states has to bring the circuit into a different state. ii) From any state, different inputs have to bring the circuit into different states.

Let a_i bring the circuit from the state s_j into the state s_k and let this be denoted by $s_k = s_j * a_i$. The state transition matrix l_{ij} defined by $l_{ij} = k$ is a Latin square. This is so because by i) no row contains an element twice and by ii) the same holds for the columns. Hence in each row and in each column every symbol (state) occurs just once. The equation $s_k = s_j * x$ is therefore uniquely solvable. Representing the states and the inputs by the same set of m symbols gives an algebraic structure which is known as a quasi group (see 16 p.7). A quasi group is a set Q in which a binary operation $\times$ is defined such that the equations $a \times x = b$ and $x \times a = b$ are both uniquely solvable for x if $a, b \in Q$. Relatively little is known about Latin squares or quasi groups. Of importance is the known (see 14). Theorem 1.3.1 A quasi group in which the associative law $a \times (b \times c) = (a \times b) \times c$ holds is a group. As a matter of fact this may be taken as the definition of a group, in which case it is of course no theorem. It is possible to define a crooked m check by means of a quasi group $(Q, \times)$ as follows:

i) Choose two elements c_0 and c_n arbitrarily in Q.

ii) Define c_i for $0 \leq i < n-1$ recursively with $c_{i+1} = c_i \times a_{i+1}$, where a_i is the i-th digit of an (n-1)-digit code word with symbols from Q.

iii) The solution of $c_n = c_{n-1} \times x$ is the check digit a_n. The check

equation of such a crooked m check would be $(\ldots((c_0 \times a_1)\times a_2)\times\ldots)\times a_n = c_n$.

Theorem 1.3.2 Every crooked m check is E_1-proof.

Proof: Let $a_1 a_2 \ldots a_n$ be a word of the code satisfying $(\ldots((c_0 \times a_1)\times\ldots)\times a_i)\times\ldots)\times a_n = c_n$ and let the i-th digit, after making a single error, be a_i'. If this erroneous word also belonged to the code then $(\ldots((c_0 \times a_1)\times\ldots)\times a_i')\times\ldots)\times a_n = c_n$ would hold and hence $(\ldots(c_{i-1} \times a_i)\times\ldots)\times a_n = (\ldots(c_{i-1} \times a_i')\times\ldots)\times a_n$. By cancelling all unaffected a's on the right of a_i and a_i', it would follow that $c_{i-1} \times a_i = c_{i-1} \times a_i'$ and after cancelling c_{i-1} from the left that $a_i = a_i'$, contrary to the hypothesis that a single error was made.

Since the vast majority of the real life errors is affecting only single digits, codes which are not E_1-proof are of little interest for the applications. The codes published until now are mostly of the crooked sum type, or at least can be viewed as such. In fact the straight sum check is also a special case, based on the cyclic group. For the decimal codes, which are after all the subject of this monograph, it is of interest to know how many Latin squares exist. This number seems to be unknown up to now, but it is at least 6×10^{27}.

It is hardly surprising that not all these possibilities have been tested on their detecting merits, especially so since most of the codes are very hard to analyse. Only two of the quasi groups of order 10 are associative, and thus admitt, as will be seen later on, a fairly easy study.

The next stage of complexity is that the way of counting is not only crooked but also dependent on the position of the digit which is fed into the circuit. Let n quasi groups be given, all based on the same set Q but with different operations $\times_i$ for $0 \leq i < n$. The recipe for making a check digit is the same as above except that the recurrence is now defined by: $c_{i+1} = c_i \times_i a_{i+1}$ for $0 \leq i < n-1$ and the check digit a_n by the equation $c_n = c_{n-1} \times_{n-1} a_n$. These codes are also E_1-proof as can be seen by the same arguments as used for the proof of theorem 1.3.2. The number of possible decimal codes becomes now hopefully or distressingly high. Hopefully because the chance that a desirable one exists is gone up, but distressing because the chance that such a one can be found gets down. The latter is even more so since the job of

testing each case gets harder too. If a periodic sequence of quasi groups is taken the procedure is less complicated and hence more manageable. Most of the new codes of chapter 3 fall into this category. Let this period be two, then the recurrence becomes:
$c_{2i+1}=c_{2i}\times_0 a_{2i+1}$; $c_{2i+2}=c_{2i+1}\times_1 a_{2i+2}$ or by taking two steps at the time $c_{2i+2}=(c_{2i}\times_0 a_{2i+1})\times_1 a_{2i+2}$. The latter relations can be considered as a ternary operation, which written in the functional notation, looks like $c_{2i+2}=g(c_{2i},a_{2i+1},a_{2i+2})$. The ternary function g is equivalent with a Latin cube of a special type, namely one constructed with the aid of two Latin squares. The code would work just as well if it were made with a more general Latin cube. That these exist is shown in the next theorem.

Theorem 1.3.2 There exists a Latin cube not based on two Latin squares.
Proof: Consider the $4\times 4\times 4$ Latin cube with the following four layers

0	1	2	3
2	0	3	1
1	3	0	2
3	2	1	0

1	0	3	2
0	1	2	3
3	2	1	0
2	3	0	1

2	3	0	1
3	2	1	0
0	1	2	3
1	0	3	2

3	2	1	0
1	3	0	2
2	0	3	1
0	1	2	3.

If this cube g(i,j,k) were based on the Latin squares p(i,j) and q(i,j) then g(i,j,k)=p(i,q(j,k)) would hold. Now if g(i,j',k')= g(i,j,k) it follows that q(j',k')=q(j,k) and hence that g(i',j',k')= g(i',j,k) are true. In the example however g(1,0,1)=0=g(1,1,0) but g(0,0,1)=1 and g(0,1,0)=2. Since g(0,0,0)=g(1,0,1)=0 and g(0,1,0)=2 but g(1,1,1)=1 it follows that g(i,j,k)=p(j,q(i,k)) does not hold either. Nor does g(i,j,k)=p(k,q(i,j)) because g(1,0,1)=g(0,1,1)=0 and g(1,0,0)=1 but g(0,1,0)=2.

A Latin cube like the one mentioned above will be called irreducible. Codes based on irreducible Latin cubes have not yet come to the attention of the author. In general an n-digit E_1-proof code can be considered as a Latin hypercube with n dimensions. An n dimensional Latin hypercube is said to be product of two Latin hypercubes if $g(i_1,\ldots,i_n)=p(i_1,\ldots,i_k,q(i_{k+1},\ldots,i_n))$. If such is the case the hypercube g is called reducible. The coordinates do not have to occur

in the same sequence on both sides of the equation. The factors of a reducible hypercube may be reducible too. Thus a n-dimensional hypercube may be the product of n-1 Latin squares. If such is the case it will be called completely reducible. All but one of the known codes which will be presented in the next chapter are completely reducible Latin hypercubes. As such they all admitt a simple graphical respresentation which consists of a "staircase" of Latin squares as shown on page 37. The check digit belonging to the code word $a_1 a_2 \ldots a_6$ is found as follows: First select the top entry a_1 then take a_2 in the column headed by a_1. After that a_3 is searched in the same row as a_2 but in the next square, a_4 is found in the same column as a_3 but in the square below and so on Finally the check digit is found at the righthand side of the last square in the same row as a_6. In the example the check digit of 671465 is found to be 1. The idea of the Latin staircase is probable very old. It can be found already in the papers of W. Friedman the renown American cryptanalyst (13). The method is good for field use and for instructional purposes.

1.4. Check equations.

In general it is advantageous not to use the functional relation between the check digit and the information digits in its explicit form $a_n = f(a_1, \ldots, a_{n-1})$, but to use an implicit form $g(a_1, \ldots, a_n)$=constant instead. Because of theorem 1.3.0 the two forms are equivalent for E_1-proof codes with one digit redundancy. For codes with a higher redundancy however the check equation is more general. The popular modulo 11 check for decimal codes is an example. This check will be discussed at length in chapter 2. Here it is sufficient to note that this code in its most common form is defined as the set C of all words satisfying the equation $\sum_{i=1}^{n} (-1)^i a_i = 0 \pmod{11}$. Now a_n, or any other a_i, is not always solvable from the equation, since $-(-1)^n \sum_{i=1}^{n-1} (-1)^i a_i$ may have the value 10, so that no decimal digit a_n can satisfy the equation. The problem might be solved by narrowing down the range of the function $f(a_1, \ldots, a_{n-1})$, in other words by taking a function for which only "proper" values for the arguments are allowed.

0	1	2	3	4	5	⑥	7	8	9										
0	4	5	1	2	3	6	7	8	9	0	1	2	3	4	5	6	7	8	9
1	8	9	0	6	7	2	3	4	5	4	7	8	9	0	6	5	1	2	3
2	7	8	9	0	6	3	4	5	1	5	6	7	8	9	0	1	2	3	4
3	6	7	8	9	0	4	5	1	2	1	0	6	7	8	9	2	3	4	5
4	0	6	7	8	9	5	1	2	3	2	9	0	6	7	8	3	4	5	1
5	9	0	6	7	8	1	2	3	4	3	8	9	0	6	7	4	5	1	2
6	3	4	5	1	2	⑦	8	9	0	6	5	①	2	3	4	7	8	9	0
7	2	3	4	5	1	8	9	0	6	7	4	5	1	2	3	8	9	0	6
8	1	2	3	4	5	9	0	6	7	8	3	4	5	1	2	9	0	6	7
9	5	1	2	3	4	0	6	7	8	9	2	3	4	5	1	0	6	7	8

0	4	5	1	2	3	6	7	8	9	0	1	2	3	4	5	6	7	8	9
1	8	9	0	6	7	2	3	4	5	4	7	8	9	0	6	5	1	2	3
2	7	8	9	0	6	3	4	5	1	5	6	7	8	9	0	1	2	3	4
3	6	7	8	9	0	4	5	1	2	1	0	6	7	8	9	2	3	4	5
4	0	6	7	8	9	5	1	2	3	2	9	0	6	7	8	3	4	5	1
5	9	0	6	7	8	1	2	3	4	3	8	9	0	6	7	4	5	1	2
6	3	④	5	1	2	7	8	9	0	⑥	5	1	2	3	4	7	8	9	0
7	2	3	4	5	1	8	9	0	6	7	4	5	1	2	3	8	9	0	6
8	1	2	3	4	5	9	0	6	7	8	3	4	5	1	2	9	0	6	7
9	5	1	2	3	4	0	6	7	8	9	2	3	4	5	1	0	6	7	8

0	4	5	1	2	3	6	7	8	9	8
1	8	9	0	6	7	2	3	4	5	2
2	7	8	9	0	6	3	4	5	1	3
3	6	7	8	9	0	4	5	1	2	4
4	0	6	7	8	9	5	1	2	3	5
⑤	9	0	6	7	8	1	2	3	4	①
6	3	4	5	1	2	7	8	9	0	9
7	2	3	4	5	1	8	9	0	6	0
8	1	2	3	4	5	9	0	6	7	6
9	5	1	2	3	4	0	6	7	8	7

Another advantage of the check equation is that it often admitts an easier analysis of the various detecting properties, since the check digit no longer plays a special role. Also from a technical point of view it is as a rule better to have a check procedure which is uniform for all digits. For general codes which are irreducible Latin hypercubes the difference would vanish. It then is only a different way of looking at the same thing.

1.5. A curious 3 digit decimal code.

It is remarkable that up to now no pure decimal codes, with a redundancy of one digit are known, which detect all single errors, all transpositions and all twin errors. It will be hard to prove that such codes cannot exist, since the proof would have to depend on special properties of the number 10, as for other number bases there do exist codes with said properties. An example will be given of a 3-digit decimal code which detects not only the error-types mentioned above, but also the jump transpositions and the jump twin errors, as well as the phonetic errors. This example shows that the non-existence would only be valid for codes with more than 3 digits. The three digit code is equivalent with a Latin square, (i.e. single error-proof), with certain special properties. Denote the elements of the square by a_{ij}. The detection of the transpositions requires that: 1) $a_{ij} \neq a_{ji}$, for $i \neq j$ and 2) if $a_{ij}=k$ then $a_{ik} \neq j$. The twin error detection requires that: 3) $a_{ii} \neq a_{jj}$, for $i \neq j$ and that 4) if $a_{ij}=j$ then $a_{ik} \neq k$ for $k \neq j$. Finally the detection of the jump transpositions requires that: 5) if $a_{ij}=k$ then $a_{kj} \neq i$, for $i \neq k$, whereas for the detection of the jump twin errors it is necessary that: 6) if $a_{ij}=i$ then $a_{kj} \neq k$, for $k \neq i$.
4) is equivalent with the condition that each row has, as a permutation of the column entries, at most one fixed point. Since each column contains all 10 decimals, every entry is fixed in some row and never in two rows. Hence each row permutation has to have exactly one fixed point. The same conclusion holds for the column permutations with regard to 6). The condition 3) requires that the main diagonal

of the square is a permutation of the decimal digits. By putting $a_{ii}=i$ all three conditions are fulfilled. This takes care of the main diagonal. The remaining 90 places outside the main diagonal can be divided into 30 triplets satisfying $a_{ij}=k$, $a_{jk}=i$, $a_{ki}=j$ with $i\neq j$, $j\neq k$ and $k\neq i$. All triplets, considered as unordered triplets, should be different. The conditions 1),2) and 5) are fulfilled if the triplets can be arranged in 30 blocks, each containing 3 decimals and each decimal occuring in 9 blocks. Moreover each pair has to occur only once as an ordered pair. The design on the next page fulfils the requirements. Each of the 30 blocks has to be oriented to define the ordered pairs. There are 16 ways to assign the orientation, since the blocks (rows) fall apart into four orientation independent classes, namely {0:3} ; {4:21} ; {22,24,26,28} ; {23,25,27,29} . The orientation can per class be inverted, independent of the other classes. An inversion of all classes results in a reflexion of the entire square, with respect to the main diagonal. Hence 8 different solutions are obtained in this way. The resulting square, corresponding with the orientation given to the right of the blockdesign, is written out below. The code does not detect all phonetic errors, but by interchanging 1 and 4, it does. The square obtained after carrying out this exchange is given next to the original one. From a practical point of view the code is perhaps not recommendable since none of the triple transcription errors aaa →bbb is detected. A further disadvantage is that none of the cyclic errors abc →bca is detected. This error-type might very well be expected for the small 3-digit code words.

The block design:

	0	1	2	3	4	5	6	7	8	9	
00	1	1	1	0	0	0	0	0	0	0	+
01	1	1	0	1	0	0	0	0	0	0	-
02	1	0	1	1	0	0	0	0	0	0	+
03	0	1	1	1	0	0	0	0	0	0	-
04	1	0	0	0	1	1	0	0	0	0	+
05	1	0	0	0	0	1	1	0	0	0	+
06	1	0	0	0	0	0	1	1	0	0	+
07	1	0	0	0	0	0	0	1	1	0	+
08	1	0	0	0	0	0	0	0	1	1	+
09	1	0	0	0	1	0	0	0	0	1	-
10	0	1	0	0	1	0	0	1	0	0	+
11	0	1	0	0	0	1	0	0	1	0	-
12	0	1	0	0	0	0	1	0	0	1	+
13	0	1	0	0	1	1	0	0	0	0	-
14	0	1	0	0	0	0	1	1	0	0	-
15	0	1	0	0	0	0	0	0	1	1	-
16	0	0	1	0	1	0	0	1	0	0	-
17	0	0	1	0	0	1	0	0	1	0	+
18	0	0	1	0	0	0	1	0	0	1	-
19	0	0	1	0	0	1	1	0	0	0	-
20	0	0	1	0	0	0	0	1	1	0	-
21	0	0	1	0	1	0	0	0	0	1	+
22	0	0	0	1	1	0	1	0	0	0	+
23	0	0	0	1	0	1	0	1	0	0	+
24	0	0	0	1	0	0	1	0	1	0	+
25	0	0	0	1	0	0	0	1	0	1	+
26	0	0	0	1	1	0	0	0	1	0	-
27	0	0	0	1	0	1	0	0	0	1	-
28	0	0	0	0	1	0	1	0	1	0	-
29	0	0	0	0	0	1	0	1	0	1	-

The resulting square:

	0	1	2	3	4	5	6	7	8	9
0	0	5	3	4	2	6	7	8	9	1
1	0	1	7	8	5	0	3	4	6	2
2	4	9	2	0	3	8	5	1	7	6
3	2	6	4	3	0	7	8	9	1	5
4	3	7	0	2	4	1	9	6	5	8
5	1	4	6	9	8	5	0	3	2	7
6	5	8	9	1	7	2	6	0	3	4
7	6	2	8	5	1	9	4	7	0	3
8	7	3	5	6	9	4	1	2	8	0
9	8	0	1	7	6	3	2	5	4	9

The interchanged square:

	0	1	2	3	4	5	6	7	8	9
0	0	2	3	1	5	6	7	8	9	4
1	3	1	0	2	7	4	9	6	5	8
2	1	3	2	0	9	8	5	4	7	6
3	2	0	1	3	6	7	8	9	4	5
4	9	5	7	8	4	0	3	1	6	2
5	4	8	6	9	1	5	0	3	2	7
6	5	7	9	4	8	2	6	0	3	1
7	6	4	8	5	2	9	1	7	0	3
8	7	9	5	6	3	1	4	2	8	0
9	8	6	4	7	0	3	2	5	1	9

This vulnerability for new error types is again an example of the designers dilemma, that the design constructed by virtue of some

regularity, is weak because of that very regularity. The irregular designs however, though often more numerous, are as a rule harder to find. Moreover the verification of the properties usually is also more difficult. For curiosities sake, an irregular code which has the same virtues as the regular one above, will be given. Though none of the triple transcription errors are detected by this code, it turns out that 82.8% of the cyclic errors is detected, which makes the irregular code superior to the regular one.

The irregular square:

	0	1	2	3	4	5	6	7	8	9
0	0	9	8	1	2	6	4	5	7	3
1	2	1	3	5	9	0	7	8	4	6
2	5	4	2	8	3	9	1	0	6	7
3	7	0	9	3	6	1	5	4	2	8
4	8	6	0	7	4	3	9	2	5	1
5	3	8	6	4	7	5	0	1	9	2
6	9	2	7	0	8	4	6	3	1	5
7	6	5	4	9	1	2	8	7	3	0
8	1	3	5	6	0	7	2	9	8	4
9	4	7	1	2	5	8	3	6	0	9

To extend such a 3-digit code to a 4-digit one is a tremendous task. It would be equivalent with the construction of a Latin cube satisfying a number of asymmetry conditions.

1.6. Single error correcting decimal codes.

According to the upper bound given in theorem 1.2.2 with e=1, the maximum number of code words in a minimum distance 3 m-ary code with n digits, is: $m^n/(1+(m-1)\times n)$. This means that for n=1 the upper bound is 1, giving the notorious, perfect 1-word code. For n=m+1 a perfect code with 2 check digits and m-1 information digits seems possible, as $1+(m-1)\times(m+1)=m^2$. It is well-known (see 37) that these codes exist if m is the power of a prime. It is not known (at least not to the author) whether these codes exist for other m. Some light throws the next theorem:

Theorem 1.6.0. If a minimal distance 3 m-ary code with m+1 digits and m^{m-1} words exists, then there exists a Graeco - Latin square of the m-th order.

Proof: Consider the subset of the code words ending with m-3 fixed digits, say zero's on the last m-3 places. This is a set of m^2 words which differ from each other on at least 3 places of the first 4. From this it follws that on the first 2 places all m^2 combinations occur exactly once. Let these digits be denoted by i and j and let the digit on the third and the fourth place be a_{ij} and b_{ij} respectively. The matrices a_{ij} and b_{ij} are the orthogonal Latin squares required to prove the theorem.

Ten years ago it would have been conjectured that this theorem disproved the existence of such a perfect code for m=10, but now only the case m=6 can be discarded. For the existence of 10x10 Graeco-Latin squares see Ryser (42) chapter 7. Such a Graeco-Latin square forms a 4 digit minimum distance 3 decimal code with 100 words. Even the existance of a 5 digit minimum distance 3 decimal code with 1000 words is unknown. The existence of the perfect code with m^{m-1} words is an interesting combinatorial problem. Its place among the other problems like the existence of finite projective planes is not yet clear. Consider the statements:

A:m is a power of a prime number.

B:There exists a field with m elements.

C:There exists a finite projective geometry with m+1 points on each line.

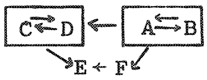
D:There exist m-1 mutually orthogonal $m \times m$ Latin squares.

E:There exists a pair of orthogonal Latin squares, i.e. a Graeco-Latin square, of the m-th order.

F:There exists a (perfect) minimum distance 3 m-ary code with m^{m-1} words of m+1 digits.

The following implications are known:(see diagram on the next page) $A \rightarrow B$; $B \rightarrow C$; $C \rightarrow D$; $D \rightarrow E$; and $F \rightarrow E$; $D \rightarrow C$; $B \rightarrow A$; and $B \rightarrow F$. The author could not prove $C \rightarrow F$ nor $D \rightarrow F$. For m=6 E has been disproved and

hence F and all others. For m=10, A and B are not true, E is true and C,D and F are open.



For practical applications a double modulo 11 check with error correcting capacity is available, with all the disadvantages of the modulo 11 method.

Chapter 2. Survey of old and new error detecting decimal codes

2.0 Introduction

In this chapter some known decimal error detecting codes will be described. Also a few new ones, designed by the author, are included. The codes are compared on the basis of certain conditions, set forth in section 2.1. In judging the codes it should be borne in mind that at the time that these codes were designed these requirements were often not known. Or more precisely that the designers were not aware that those criteria were of importance. Two examples will suffice to make this point clear. The I.B.M. code of section 2.3.0 was designed to detect the single errors as well as the transpositions. The code does detect the single errors for 100%, but the transpositions for only 98%, since the transposition of 0 and 9 escapes detection. In some applications therefore the code words in which a 0 and a 9 occur on adjacent positions, are omitted (see 33). A more serious flaw however is that this code does not detect the jump transpositions at all, a flaw which could have been overcome relatively easy. A second example is the biquinary code of section 2.3.1 This code was designed with the same objective as the I.B.M. code, as a matter of fact the purpose was to do better on the transpositions. The code was a success in the sense that the detection rate of the transpositions is 100%, but it was sheer luck that the jump transpositions did not escape detection entirely. That the new codes of the section 2.3 are doing better, is therefore partly due to the fact that they are taylored for the requirements set forth in section 2.1. More convincing are therefore the tests on the set of 12112 real life errors drawn from the daily operations of a clearing institute. All these errors are made in code numbers with 6 decimals. The errors of forgotten decimals have been eliminated beforehand.

2.1 The requirements for decimal codes

It has already been stressed before that the requirements for a good code cannot be set absolutely. They depend on the type of equipment

used and also on (the knowledge of) the error habits of the human beings involved. The methods explained in the chapters 3, 4 and 5 are however also applicable to many other requirements than the ones which follow:

1) The single errors are considered to be most important. The weight of 800 points is given to these errors.
2) Next come the transpositions (of adjacent digits) with a weight of 100 points.
3) The twin errors, aa→bb, get 10 points, just as
4) the jump transpositions, abc→cba.
5) The separated twin errors, aba→cbc, is a less important class getting 5 points. These errors will be called jump twin errors.
6) The phonetic errors, 13→30 and the like, also get 5 points. These errors may be for some languages of little importance, but this is only an opinion, since no corroborative evidence is available.

In the next section the detection rates are often given without a proof, since these will be given, except for the trivial cases, in the chapters 3, 4 and 5. The mathematical formulation of the requirements is also postponed, since such a formulation is dependent on the method employed. All but one of the E_1-proof codes of the next section are of the completely reducible type. As such they form a set of words satisfying the recursion $c_i = c_{i-1} \times_{i-1} a_i$, where c_0 and c_n are fixed decimals, a_i are the digits of the word and where the decimals form a quasi group with respect to each of the operators $\times_i$. A burst of two errors on the positions i and i + 1 is detected if and only if $(c_{i-1} \times_{i-1} a_i) \times_i a_{i+1} \neq (c_{i-1} \times_{i-1} a_i') \times_i a_{i+1}'$ no matter what value c_{i-1} has. This inequality cannot be true for all possible bursts, for if a_i, a_{i+1} and a_i' are given, there always exists an a_{i+1}' such that the equality holds. That the requirements for detecting at the same time all transpositions, all twin errors and all jump transpositions are not per se too heavy, is shown by the example in section 1.5.

For decimal codes based on a group, with an operation denoted by +, a general form of the operators $\times_i$ may be defined by $a \times_i b = a + f_i(b)$. If $f_i(x) = x$ for all x and for all i then the code is a straight check.

If f_i is an automorphism of the group then the code is called a weighted check. This is reasonable because the automorphisms satisfy the relation: $f(x + y) = f(x) + f(y)$ and therefore behave like a weight. The weights are, just like the automorphisms, linear operators and they are therefore often written as a multiplier;$f(x)=fx$.
The permutations form also a group, the symmetric group. In the symmetric group the automorphisms form a subgroup. The product of two permutations f and g is defined as the permutation which brings x into the element $f(g(x))$, it is denoted by fg. Hence $fg(x) = f(g(x))$ holds (see 55). If $f_i = ff_{i-1}$ for a fixed permutation f the code is called progressive. Progressive codes are periodic. If the period is 2 then the code is called alternating. In an alternating code the operations a+b and a+f(b) are applied alternatively. A weighted alternating code however may also be considered as a straight code in which the same operation is used. throughout. This operation is defined by $a\times b = f(a)+b$.. The same remark holds for all weighted progressive codes, with the same definition for the single operator. This construction is merely a version of the algorithm of Horner. It is an illustration of the fact that some of the properties defined above do not exclude each other, and are thus not suitable for a classification.
The discussions above are valid for all groups of order 10. It is well-known from the theory of groups that there are 2 groups of order 10 available (see Hall, 16, p. 52) i.e. the cyclic group of order 10 and the dihedral group of order 10. The first one is the group of the rotations of a regular 10-gon, denoted by C_{10}. Its operation is also the addition modulo 10 or what is the same, the addition in a cyclic 10-counter. The cyclic groups are abelian, that is the commutative law $a\times b = b\times a$ holds. The second group is the dihedral group, that is the group of the transformations of the pentagon. This group contains not only the rotations, but also the reflexions. It is denoted by D_5. The group is not commutative, for rotating the pentagon over say 72 degrees and then reflecting it, is not the same as first reflecting it and then rotating it over 72 degrees. The idea of applying groups is, that the condition for the error detection is simplified by virtue of the associative law. As will

be seen in chapter 3, the use of the cyclic group gives a still greater simplification

Much of the effort, spent on the construction of decimal codes centers on the detection of the transposition errors. This typical human type of error has been bothering the cryptologists for a long time, it is mentioned by Friedman as a psychological lapsus calami as early as 1932 (see 13). Since the straight modulo 10 check obviously is insensitive for transpositions and in view of the fact that the altermating sum, modulo an odd number, met the requirement, one naturally tried to do something like that for the decimal codes too. The I B.M. code of 2.3.0 seems to be the first trial in that direction Unfortunately the detection appeared not to be flawless and when some authors proved, that no decimal E_1-proof code with one redundant decimal could be transposition-proof, the codes based on the eleven check became very popular. Even nowadays many people still believe that one has to use modulo 11 checks for the detection of the transpositions. Actually the non-existence proof mentioned above is only valid for codes based on the cyclic group C_{10}, with generalized weights which are independent of the words itself. The codes of 2.3.4 on page 56 are examples of codes designed for detecting transpositions which are unfortunately no longer E_1-proof.

The biquinary codes of chapter 5 designed by the present author are however both E_1- and transposition-proof. The first one has many faces. It is an alternating code in the sense that it can be defined by applying two quasi groups alternatively. It has also an interpretation as a biquinary code and as a code based on addition modulo 10 with weights and checkvalue (c_n) depending on the value of the digits Last but not least the code can be interpreted as a code based on the dihedral group with generalized weights and as such it would fall under 2.3.2 The generalized biquinary code however loses this interpretation (see chapter 5). The merit of the generalized biquinary code is that it does much better in the detection of the twin errors and the jump transpositions. Finally the application of the dihedral group D_5 turns out to give scores of E_1- and transposition-proof codes. Some of these do

even better than all other decimal codes mentioned above (see chapter 4). Up to now no decimal code, scoring 100% in all the categories, has come to the attention of the author. He was also unable to prove that such a code does not exist. Such a proof would have to depend on properties of the number 10 since for other number bases codes like that do exist. Moreover in section 1.5 an example of a 3-digit decimal code scoring 100% in all 6 categories, has been given. The performance of the modulo 11 check is difficult to compare with that of the pure decimal codes since their redundancy is higher. For that reason alone these codes detect 1% more in the realm of the random errors. A 6-digit decimal code satisfying a modulo 11 check equation has at most 90910 words whereas a pure decimal code will have 100000 words. Decimal codes applied in situations where a modulo 11 check could also be used, have therefore a hidden redundancy, which is not taken into account in the performance comparison tables. The main disadvantage of the eleven checks is that the lexicographical ordering, according to the information digits, of the code shows gaps, in contradistinction to a pure decimal code. This may be a disadvantage for the efficiency of the file-handling and storage, but when it comes to application in an existing system it implies a recoding of about 10% of the code words. The resulting inconvenience for customers and the potential danger for more errors makes the application of the modulo 11 check in those cases very unattractive. In some applications this difficulty is overcome by giving no check digit in those cases where the 10-th symbol would be required. The blank is then playing the role of the 10-th symbol. The drawback is that the code will no longer have a fixed length. In other applications the 0 has to play the double role of the 0 and the 10-th symbol. However the remedy is worse than the disease, as the code stops being E_1-proof.

2.2 A classification of decimal codes

The decimal codes, discussed in this chapter, may be classified as follows:

1) The non-E_1-proof codes.

1.1) Codes modulo k, with 10 > k.

1.2) Codes modulo 10, using weights divisable by 2 or 5.

1.3) The Bull type codes, using the sum of 2 alternating sums, each with a different modulus smaller than 10.

1.4) Various modulo 11 checks, in which the "0" and the "10" are identified.

2) The E_1-proof codes with 1 decimal redundancy. All but one of the codes, mentioned here, are of the completely reducible type and as such they can be defined by the Latin staircase method. In the text however other definitions, admitting an easier analysis, will be employed.

2.1) Codes based on the addition modulo 10, i.e. the cyclic group C_{10}.

2.1.1) The straight sum check.

2.1.2) The alternating sum check.

2.1.3) The weighted sum checks.

2.1.4) Sum checks with generalized weights.

2.2) Biquinary codes.

2.2.1) Alternating biquinary codes.

2.2.2) Generalized biquinary codes.

2.3) Codes based on the multiplication in the dihedral group of the pentagon, i.e. D_5.

2.3.1) Straight product check.

2.3.2) "Weighted" product checks.

2.3.3) Periodic product checks with generalized weights.

2.3.4) Non-periodic product checks with generalized weights.

3) The E_1-proof codes with a higher redundancy than one decimal.

3.1) Checks based on the addition modulo k, with k > 10.

3.1.1) Various modulo 11 checks.

3.1.2) Checks modulo k, with k > 11.

2.3 Description of the various error detecting decimal codes

2.3.0 Checks modulo 10

The straightforward generalization of the parity check is the straight modulo 10 check. This code consists of all words satisfying $a_1 + a_2 + \ldots + a_n = c \pmod{10}$. Although the code detects all single errors (see 1.2), the obvious disadvantage is that no transpositions are detected. The twin errors, $aa \to bb$, are caught for 88.9%, since $2a = 2b \pmod{10}$ if $a = b + 5 \pmod{10}$. It is a poor consolation that the phonetic errors are detected for 100%.

A considerable improvement gives the alternating check modulo 10. The check equation for the alternating check is: $a_1 - a_2 + a_3 - \ldots = c \pmod{10}$. This check detects 8 out of the 9 transpositions, since $a_1 - a_2 = a_2 - a_1 \pmod{10}$ only holds if $2a_1 = 2a_2 \pmod{10}$ or equivalently $a_1 = a_2 + 5 \pmod{10}$. The jump transpositions still remain undetected. A major drawback is that the twin errors now escape detection completely, since $a - a = b - b$ for all a and b. The difficulty is clearly that 10 contains a factor 2 and in fact for an odd modulus this type of check would detect all transpositions. The alternating check of the form $a_1 + 2a_2 + a_3 + 2a_4 + \ldots = c \pmod{10}$ does detect all transpositions, but is unattractive since it is not E_1-proof, as an error of 5 units, on the even positions, does not change the sum modulo 10. The root of the trouble is that the function: $2x \pmod{10}$ has always an even value.

The I.B.M. code is an intelligent trial to improve this situation, by defining a permutation f by: $f(x) = \begin{cases} 2x & \text{if } 2x < 10 \\ 2x-9 & \text{if } 2x \geq 10 \end{cases}$ (the carry is added to the product $2x \pmod{10}$), hence $f = \begin{Bmatrix} 0123456789 \\ 0246813579 \end{Bmatrix}$. The said I.B.M code consists of all words satisfying: $a_n + f(a_{n-1}) + a_{n-2} + f(a_{n-3}) + \ldots = c \pmod{10}$. This code was a big stride forward, but it did not detect the transpositions completely, as the transposition of 0 and 9 goes by unnoticed, giving a detection rate of 97.8%. Because of its alternating character none of the jump transpositions is detected. As will be shown in chapter 3, it is not accidental that 1 out of the 45 possible transpositions remains undetected by codes using fixed permutations in combination with the

addition modulo 10.

It is however possible to do better with respect to the jump transpositions, by using sequences of (generalized) weights with a higher period than 2. It is well-known from number theory that the powers of an arbitrary number modulo n, form a periodic sequence. The number 9 (= -1 (mod 10)) has for instance the period 2 modulo 10, as $9^2 = 81$ and 81 = 1 (mod 10). The period of 3 modulo 10 is 4, as can be easily verified. The code defined by: $\Sigma\ 3^i a_i = a_1 + 3a_2 + 9a_3 + 7a_4 + a_5 + \ldots =$ $= c$ (mod 10) besides being E_1-proof, detects 8 out of the 9 transpositions since $3^i a_i + 3^{i+1} a_{i+1} = 3^i a_{i+1} + 3^{i+1} a_i$ (mod 10) if $2a_i = 2a_{i+1}$ (mod 10). The jump transpositions are also detected for 88.9% since $3^i a_i + 3^{i+2} a_{i+2} = 3^i a_{i+2} + 3^{i+2} a_i$ (mod 10) leads to $8a_i = 8a_{i+2}$ (mod 10). Also the twin errors are detected for 88.9% as $3^i a + 3^{i+1} a = 3^i b + 3^{i+1} b$ (mod 10) is equivalent with $4a = 4b$ (mod 10). But unfortunately now the jump twin errors give trouble, as $3^i a + 3^{i+2} a = 3\ (1+9)a = 0$ (mod 10) for all a.

It is of course not necessary that the weights form a geometric progression modulo 10 and one sometimes sees a weighted code, defined by $a_1 + 3a_2 + 7a_3 + a_4 + 3a_5 + \ldots = c$ (mod 10). This code, which is of period 3, is equally good on the single errors and the transpositions, but does better on the jump twin errors than the former one did. Of the jump twin errors it detects 88.9% on 2 out of the 3 positions and 0% on the third, giving a nett result of 59.3%. The drawback is that the same rate now holds for the twin errors, instead of the 88.9%.

Since 1, 3, 7, 9 are the only proper weights which are admissable in view of the complete detection of the single errors and since virtually all possible combinations are tried it is reasonable to turn the attention to codes with generalized weights. An obvious improvement of the I.B.M. code is to make its period higher by using powers of the permutation f for the successive weights. This generalization gives a code defined by: $a_n + f(a_{n-1}) + f^2(a_{n-2}) + f^3(a_{n-3}) + \ldots = c$ (mod 10). The code which is obviously E_1-proof, still has a detection rate of 97.8% for the transpositions. The detection of the jump transpositions

depends on how often $f^i(a) + f^{i+2}(b) = f^i(b) + f^{i+2}(a) \pmod{10}$ holds for $a \neq b$. Put $A = f^i(a)$ and $B = f^i(b)$, then the condition becomes: $A + f(B) = B + f(A) \pmod{10}$ or $A - f(A) = B - f(B)$. The function $X - f(X)$ has the values 0-0, 1-4, 2-8, 3-3, 4-7, 5-2, 6-6, 7-1, 8-5 and 9-9 or modulo 10: 0, 7, 4, 0, 7, 3, 0, 6, 3, 0, so that 8 out of the 45 combinations a, b fulfill the equation and hence 82.2% of the jump transpositions will be detected. In a similar way the detection rate of the twin errors is found to be 93.3% and for the jump twin errors 95.6%. The phonetic errors have a detection rate of 89.6%. In chapter 3 it will be shown that the permutation g defined by $g(x) = f(x) + 6$, or $g = \begin{Bmatrix}0123456789\\6802479135\end{Bmatrix}$ has the same rate of detection for the single errors, the transpositions and the twin errors as f has. On the other error types the code defined by: $\Sigma\, g^i(a_{n-i}) = c \pmod{10}$ is better than the one defined with f. It detects the jump transpositions and the jump twin errors both for 95.6% and the phonetic errors for 90.3%. An oculist from the Leiden University, Dr. A.D. Colenbrander, who needed an error detecting code for a hostpital administration was not satisfied with the codes known to him and designed an interesting and remarkably good one as follows: The 10 non-zero residue classes modulo 11 form a group under multiplication. In particular multiplication by 2 modulo 11 gives a permutation of these 10 classes. Coding the class 10 by 0 and the classes 1, 2, ..., 9 by 1, 2, ..., 9 thus gives a permutation of the decimals $f = \begin{Bmatrix}0123456789\\9246801357\end{Bmatrix}$. The code is defined by $\Sigma\, f^i(g(a_i)) = 0 \pmod{10}$, where g is an arbitrary permutation. In particular g can be chosen so that the code becomes $\Sigma\, h(i + a_i) = 0 \pmod{10}$, where $i + a_i$ is to be taken modulo 10 too. The latter code detects 100% of the single errors, 97.8% of the transpositions, 93.3% of the twin errors, 95.6% of the jump transpositions and jump twin errors and 100% of the phonetic errors. His way of making a permutation resembling multiplication by 2 is apparently more fortunate than the one of the I.B.M. code. His code is a close analogue of the "best" modulo 11 code defined by $\Sigma\, 2^i a_i = c \pmod{11}$. It is also meritorious that he uses the permutation f in a "geometric" progression. It is rather unsatisfactory

to try haphazardly some permutations and moreover it is by no means necessary to limit the generalized weights to the powers of one single permutation. In chapter 3 an exhaustive search for the most favourable combination of permutations has therefore been carried through. It turned out that theoretically the code defined by: $\Sigma f_i(a_i) = c \pmod{10}$ where the f_i's are given in section 3.5, is one of the best. This check is shown to detect 97.8% of the transpositions and of the twin errors, and 95.6% of the jump transpositions and the jump twin errors, whereas the phonetic errors are detected for 97.9%. So far the story of the codes modulo 10.

2.3.1 Biquinary codes

The first pure decimal code which is both E_1- and transposition-proof, is perhaps less powerful than the best codes described in the previous section, but it is interesting for other reasons. Its weakness lies in the rather poor detection rate for the twin errors and the jump transpositions, namely 55.5% and 66.7% respectively. The code is described at length in chapter 5 and it will suffice here to mention that the phonetic errors are detected for 100% and the jump twin errors for only 66.7%. The version by Benard, which is also described in chapter 5, has the same properties except for the twin error detection which is only 27.8%. The generalization, which is of a later date (see 5.3), scores also 100% for the single errors, the transpositions and the phonetic errors. A detection rate of 88.9% holds for the twin errors, and the jump transpositions, whereas the jump twin errors are detected for 66.7%. One of the merits of these biquinary codes is that they lend themselves to a relatively simple technical implementation.

2.3.2 The dihedral codes

In chapter 4 codes of a quite different nature are described. Instead of addition modulo 10 the multiplication in the dihedral group D_5, of the order 10, is employed. This group is non-abelian, since $a \times b = b \times a$ does not always hold true. It follows therefore that the straight product code defined by: $a_1 \times a_2 \times \ldots \times a_n = c$ in D_5, does not miss all trans-

positions. In fact 2 out of the 3 transpositions are detected. The same fraction of the twin errors, the jump transpositions and the jump twin errors is detected. The alternating product check, defined by:
$a_1 \times a_2^{-1} \times a_3 \times a_4^{-1} \times \ldots = c$ in D_5 is, in this group, no improvement, since now 5 out of the 9 transpositions and all the twin errors escape detection. Checks using an analogon of the weights are also far from satisfactory, but there are many combinations of generalized weights which do yield excellent results. It is shown in chapter 4 that 100% detection of the transpositions can be achieved in combination with 95.6% of detection for the twin errors and 94.2% for the jump transpositions and twin errors. There exists a progressive code of the form:
$f(a_1) \times f^2(a_2) \times f^3(a_3) \times \ldots = c$ in D_5 which has the qualities mentioned above and which scores 95.3% in the phonetic errors, with
$f = \begin{Bmatrix} 0123456789 \\ 1576283094 \end{Bmatrix}$. There also exist many non-progressive codes of the form: $f_1(a_1) \times f_2(a_2) \times f_3(a_3) \times \ldots = c$ in D_5, which are even phonetic error-proof. In 4.5 it is described how the permutations f_i can be constructed.
Comparing the codes of chapter 3 and chapter 4 is not quite as simple as the analysis given above suggests. A code, missing 1 out of the 45 transpositions does not necessarily miss 1/45-th of the transpositions, as the assumption that the transpositions are uniformly distributed is very unlikely. Much more about this distribution should be known in order to be able to construct better codes, which capitalize upon this fact.

2.3.3 Codes modulo k, with k > 10

The best known higher modulus codes are the ones modulo 11. The disadvantages of the modulo 11 codes in general has been discussed in the section 2.1. Here only the detecting qualities will be subjected to analysis. Not to be recommended is the straight modulo 11 check, defined by $\Sigma\, a_i = c \pmod{11}$, because it misses all transpositions. The alternating checks modulo 11, like $\Sigma\, (-1)^i a_i = c \pmod{11}$ and $a_1 + 2a_2 + a_3 + 2a_4 + \ldots =$ $= c \pmod{11}$, though better, cannot be recommended either because they still miss all jump transpositions. There are however scores of possibilities for good weighted codes modulo 11. All the non-zero weights are admissable

with respect to the detection of the single errors. Transposing the digits of the i-th and the j-th position is detected, provided that $w_i \neq w_j \pmod{11}$, where w_k is the weight of the digit on the k-th position. The twin errors on the same positions are detected if $w_i + w_j \neq 0 \pmod{11}$. The main interest is of course to find sequences of weights fulfilling the 2 requirements for $j = i + 1$ and $j = i + 2$ at least. The arithmetic progression modulo 11: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, etc. (the 0 has to be skipped) is often applied. A flaw of this choice is that the 5 and the 6 are on adjacent positions, so that not all twin errors are detected. Another slight disadvantage is that not all phonetic errors are detected, since $ix = i + (i+1)x \pmod{11}$ holds for $x = -i \pmod{11}$ and hence only for $i = 1$ all phonetic errors are detected and otherwise 7 out of the 8. The geometric progression modulo 11 of the powers of 2 is better. Not only is this code twin error-proof, but it does as by miracle detect all phonetic errors, since $2^i \cdot x + 2^{i+1} \cdot 0 = 2^i \cdot 1 + 2^{i+1} \cdot x \pmod{11}$ holds only if $x = 2x + 1 \pmod{11}$ or $x = 10 \pmod{11}$, which is impossible since $9 \geq x \geq 0$.

Not so lucky is the progressive code with $w_i = 3^i$ since $3^i x = 3^i + 3^{i+1} x \pmod{11}$ holds, if $x = 5$ holds. Beckley (see 2) denounces both the arithmetic and the geometric progression - and he recomments a progression-free set of weights. His argument is that the progressive weights are vulnerable for the type of error, like 2560004→2056004→2005604→2000564, called shift errors. For each choice of progression there are certain combinations which can be shifted freely, that is these combinations are such that all the shifts are not detected. For the Beckley choice of weights there are combinations of digits which are not detected in case of a single shift, but which are in fact detected in case a double shift occurs. On the other hand there are also pairs of digits which are detected in case of a single shift and not detected for the double shift. It is questionable whether it has an influence on the average number of undetected errors, unless one presupposes a higher frequency of use of the vulnerable combinations. The weights given by him do not quite meet the specifications since the combinations 13; 26; 39; 41;

54; 67; 82; 95 are all immune for both the single and the double shift on the positions 9 and 8, as can be verified easily. His weights are 9, 10, 7, 8, 4, 6, 3, 5, 2, 1 and $10\times3 + 7\times9 \equiv 7\times3 + 8\times9 \equiv 8\times3 + 4\times9$ (mod 11).

Codes using a modulus higher than 11 are possible in cases where a higher redundancy is admissable. If one has to protect a 7 digit code, of which only $5\cdot10^6$ words are needed, then it is perhaps advisable to employ an 8 digit code, satisfying a check equation modulo 19. By doing so it becomes possible to detect random errors for about 95% automatically at the input, instead of during the processing.

2.3.4 Codes which are not E_1-proof

There may be cases where the application of a modulus below 10 is attractive, even though these codes cannot be E_1-proof. The case of a check equation modulo 2 is of interest, since it gives rise to a code detecting all restricted single errors (i.e. single errors of 1 unit, like 6→7 or 4→3). The code may be useful for "small" sets, say less than 500 items, occurring on questionaires. In general it is a good policy to use the natural redundancy for error detection. If one has to code 1400 items one would need 4 decimal digits anyhow. By using codes with check equation modulo 7, one gets a certain protection without extending the length of the code words. As soon as one adds a check digit, it is of course inefficient to use a check modulo 7. Only one case came to the attention of the present author. The code in question is defined by all the words with the property that the decimal value is divisable by 7. Hence $7 \mid \Sigma\, 10^{i-1} a_i$ for all code words $a_n \ldots a_2 a_1$ and since $10 \equiv 3$ (mod 7) the weights w_i satisfy $w_i \equiv 3^{i-1}$ (mod 7). Single errors are not detected if $a_i \equiv a_i'$ (mod 7), that is if a_i equal 0, 1 or 2 and a_i' equals 7, 8 or 9 respectively. Assuming an uniform distribution this gives a detection rate of 1/15. The codes modulo 9 will yield (under the same assumption) a much better rate, namely 1/45, since only the error from 0 to 9 will remain unnoticed. According to the error samples mentioned in the introduction this does

not hold true since the combined frequencies of $0 \to 7$; $1 \to 8$ and $2 \to 9$ are much lower than the frequency of the transcription error $0 \to 9$. It goes without saying that the frequencies of the converse errors have also to be taken into account. Only for completeness sake some of the codes modulo 9 will be included in the comparative code charts. Up to now no codes using another group than the cyclic one seem to have been applied. It is doubtful whether much improvement can be achieved in that way. If reliable frequency tables were known, then it would certainly be possible to improve the single error detection rate considearbly. This could be done by recoding the digits so that 0 and 9 are represented by 2 symbols with a lower transition frequency.
Also for completeness sake and perhaps as a warning some non-E_1-proof codes will be discussed.
First of all one sometimes sees codes modulo 10, which use degenerated weights like 2, 4, 6, 8 or even 5. The even weights donot detect errors like $a \to a+5$, so that 1 out of the 9 single errors on that position escapes. Positions on which the weight 5 is used admit single errors for which the parity is unchanged, so that 4 out of the 9 single errors slip through. Most notorious are the codes defined by $\Sigma\, ia_i = c \pmod{10}$; $\Sigma\, 2^i a_i = c \pmod{10}$ and the codes with the weights 121212 or 1234678. Still another type of codes which cannot be recommended are the double modulus alternating codes (see 11) called Bull codes for short. These codes were originally introduced as E_1- and transposition-proof codes. They are defined as follows: Let p and q be 2 integers satisfying $11 \geq p+q > p,\ q > 2$ and let $c_p = \Sigma\, (-1)^i a_i \pmod{p}$ and $c_q = \Sigma\, (-1)^i a_i \pmod{q}$, with $p > c_p \geq 0$ and $q > c_q \geq 0$. From the assumptions it follows that $9 \geq p-1+q-1 > c_p+c_q \geq 0$ holds, so that $c_p + c_q$ can be used as a check digit. The underlying idea probably was that as soon as p or q was odd, the transpositions will be detected by one of the two equations. The combinations 3,5; 4,5; 3,7; 4,7 and 5,6 are recommended (see 11) and also 3,8 can be tried. A further analysis shows however that the claims are not justified. In another patent, codes are proposed in which not the sum, but the number $c_p + pc_q$ is used as check symbol, which is of course no longer decimal. The application of a check symbol with that many values opens the possibilities for codes

far superior, so that this variant cannot be recommended either. Finally the codes modulo 11 with a decimal check symbol with the 0 in the double role of 10 and 0, are included primarily as a warning, since this twist destroys all the good qualities which the modulo 11 codes may have.

2.4 Results of a test on life errors

Several codes have been tried on the sample of 12112 errors in 6 digit words. This sample is too small to give significant results for the better codes. This is especially so since there are a few pairs which occur with a multiplicity of about 20 and even one pair (903559→145379) with a multiplicity of 89. A second test has therefore been performed on the non-single errors after removal of all duplicates. This smaller example consists of 1665 double errors and 471 multiple errors. Only E_1-proof codes are tested on these 2136 errors. The numbers of undetected errors per check system are listed in the second table below. In the same table the mathematical expectations per 1000 errors are given for the various error-types like the transpositions, the twin errors, the jump transpositions and twin errors and the phonetic errors and finally the random errors. These expectations are based on the assumption that the various possible transpositions etc. are equally likely, which is certainly not the case in the present sample. The last two columns of the same table give the percentage of the undetected errors with respect to the non-single errors and with respect to all errors as calculated from the mathematical expectations. The check systems are listed in descending order of the number of undetected errors from the sample.

Table of test results on the 12112 pairs of 6-digits words.

Check system	Number of not detected single errors	double errors	multiple errors	total
Σa_i=c(mod 9)	621	1391	84	2096
Σa_i=c(mod 11) with 10=0	308	1395	88	1791
$\Sigma i a_i$=c(mod 10)	1650	52	70	1772
Σa_i=c(mod 10)	0	1397	73	1470
Σa_i=c(mod 11)	0	1386	80	1466
$\Sigma\ 2^i a_i$=c(mod 10)	1124	227	76	1427
Bull type 4,5	955	331	72	1358
Bull type 4,7	698	323	61	1082
Weighted 121212 modulo 10	671	157	145	973
Weighted 121212 modulo 9	621	169	182	972
Bull type 3,7	558	318	76	952
$\Sigma(-1)^i a_i$=c(mod 9)	621	250	56	927
Bull type 3,5	424	394	96	914
$\Sigma 2^i a_i$=c(mod 9)	621	115	81	817
Alternating dihedral code	0	762	54	816
Weighted 212121 modulo 10	503	142	81	726
Bull type 5,6	317	264	59	640
$\Sigma(-1)^i a_i$=c(mod 11) with 10=0	254	238	66	558
Straight dihedral code	0	454	59	513
Weighted 121212 mod.11; 10=0	186	178	88	452
Bull type 3,8	85	278	72	435
$\Sigma(-1)^i a_i$=c(mod 10)	0	347	63	410
Weighted 313131 modulo 10	0	265	135	400
Weighted 137137 modulo 10	0	236	156	392
$\Sigma 3^i a_i$=c(mod 7)	142	146	82	370
$\Sigma 3^i a_i$=c(mod 10)	0	200	147	347
$\Sigma 3^i a_i$=c(mod 11), with 10=0	192	81	60	333
$\Sigma 2^i a_i$=c(mod 11), with 10=0	185	74	64	323
$\Sigma i a_i$=c(mod 11), with 10=0	175	94	52	321
$\Sigma(-1)^i a_i$=c(mod 11)	0	217	56	273
Weighted dihedral code	0	223	50	273
Biquinary code, Benard version	0	112	134	246
Progressive dihedral code	0	54	174	228
IBM code	0	167	52	219
Weighted 212121 modulo 11	0	141	69	210
Weighted 313131 modulo 11	0	140	43	183
First biquinary code	0	138	45	183
Generalized IBM code	0	113	44	157
Generalized biquinary code	0	79	68	147
$\Sigma f_i(a_i)$=c(mod 10)	0	65	60	125
Modified generalized IBM code	0	69	48	117
$\Sigma f^i(a_i)$=c(mod 10) (Colenbrander)	0	78	34	112
Best dihedral code	0	56	50	106
$\Sigma 3^i a_i$=c(mod 11)	0	44	55	99
$\Sigma i a_i$=c(mod 11)	0	57	40	97
Weighted 463521 mod.11 (Beckley)	0	52	42	94
$\Sigma 2^i a_i$=c(mod 11)	0	38	55	93

Test on sample of 2136 non-single errors	measured frequencies			theoretical estimates of undetected errors							
check system	double errors	multiple errors	total	transpositions	twin errors	jump transpositions	jump twin errors	phonetic errors	random errors	percentage of non-single errors	percentage of all errors
$\Sigma a_i = c \pmod{10}$	1235	67	1302	1000	111	1000	111	0	100	59.3%	11.87%
$\Sigma a_i = c \pmod{11}$	1228	69	1297	1000	0	1000	0	0	91	58.2%	11.64%
Alternating dihedral code	691	52	743	556	1000	267	267	0	100	38.3%	7.66%
Straight dihedral	390	47	437	333	333	333	333	0	100	24.3%	4.86%
$\Sigma(-1)^i a_i = c \pmod{10}$	311	53	364	111	1000	1000	111	0	100	19.3%	3.87%
Weighted 313131 modulo 10	244	44	288	111	111	1000	111	0	100	14.9%	2.98%
Weighted dihedral	206	49	255	111	111	267	267	250	100	12.2%	2.45%
$\Sigma(-1)^i a_i = c \pmod{11}$	198	52	250	0	1000	1000	0	125	91	13.5%	2.70%
Weighted 137137 modulo 10	192	49	241	111	407	111	407	0	100	12.7%	2.53%
$\Sigma 3^i a_i = c \pmod{10}$	174	49	223	111	111	111	1000	0	100	12.7%	2.53%
IBM code	151	50	201	22	67	1000	111	125	100	10.5%	2.11%
Weighted 212121 modulo 11	134	66	200	0	0	1000	0	63	91	8.3%	1.67%
Weighted 313131 modulo 11	129	41	170	0	0	1000	0	125	91	8.5%	1.70%
First biquinary	126	43	169	0	444	333	333	0	100	8.2%	1.64%
Biquinary (Benard)	107	40	147	0	728	333	333	0	100	9.6%	1.93%
Generalized IBM	90	43	133	22	67	178	67	104	100	6.3%	1.20%
Generalized biquinary code	70	59	129	0	111	111	333	0	100	5.4%	1.09%
$\Sigma f_i(a_i) = c \pmod{10}$	55	55	110	22	22	44	44	27	100	5.3%	1.02%
Progressive dihedral code	51	53	104	0	44	59	59	47	100	4.3%	0.86%
$\Sigma f^i(a_i) = c \pmod{10}$ (Colenbrander)	70	33	103	22	67	44	44	0	100	5.3%	1.05%
Generalized IBM modified	62	37	99	22	67	44	44	97	100	5.5%	1.10%
$\Sigma 3^i a_i = c \pmod{11}$	44	49	93	0	0	0	0	125	91	3.5%	0.70%
$\Sigma i a_i = c \pmod{11}$	57	35	92	0	200	0	0	125	91	4.5%	0.90%
Weighted 463521 (Beckley)	49	42	91	0	0	0	250	125	91	4.1%	0.82%
Best dihedral	45	41	86	0	44	59	59	0	100	4.2%	0.83%
$\Sigma 2^i a_i = c \pmod{11}$	35	38	73	0	0	0	0	0	91	3.2%	0.64%

2.5 Conclusions

The general impression is that the tests give a good confirmation of the theory. There are a few discrepancies which show that the assumption of the uniform error distribution is invalid. One would for instance expect that the check equation $\Sigma\ 2^i a_i = c \pmod 9$ would be superior to $\Sigma\ 3^i a_i = c \pmod 7$. The first check does not detect the single errors 0→9 and 9→0, whereas the second one does not detect 0→7; 1→8; 2→9 and 7→0; 8→1; 9→2, but the latter six together have a much lower frequency than the first two. The check equation $\Sigma\ 2^i f(a_i) = c \pmod 9$, where f is a permutation such that f(0) = 0 and f(7) = 9, would yield a much better result on the single errors (17 instead of 621) than $\Sigma\ 2^i a_i = c$ (mod 9). It is however dangerous to build a code on the assumption that the transcription errors 0→7 and 7→0 are per se rare. The danger may be illustrated by the typical high frequency of the transcription errors 7→9 and 9→7, which probably arises from the phonetic resemblance of "zeven" and "negen" which is Dutch for 7 and 9. The obvious conclusion is that only the E_1-proof codes are of practical value.

Though the material is not sufficient for the ultimate choice of the "best" code, it is clear that only the lower half of the second table contains the serious candidates. It is also clear that the modulo 11 codes are by no means the only answer to the detection problem. If there are reasons for avoiding the modulo 11 codes, then there are certainly competative pure decimal codes available to the system designer. This is especially so since the modulo 11 codes look better because they profit from the hidden redundancy which they require (see section 2.1).

Chapter 3. Codes based on the cyclic group of order 10.

3.0 Some definitions.

Let D be a set with 10 elements and let + be a binary operation defined on that set such that (D,+) is a group. Consider all ordered n-tuples of elements from D, in other words the set D^n. Let f_i for $1 \leq i \leq n$, be n functions with value and argument both in D. Hence for $x \in D$ also $f_i(x) \in D$, which is denoted by $f_i \in D^D$. Let furthermore c be an arbitrarily chosen element of D and let a code C be defined as the subset of D^n consisting of the n-tuples $a_1a_2...a_n$ which satisfy: $\sum_{i=1}^{n} f_i(a_i)=c$, hence $C=\{a_1a_2...a_n \mid \sum_{i=1}^{n} f_i(a_i)=c\}$. The purpose of this chapter is to find the functions f_i which yield the "best' codes.

A function which maps D onto D is called a permutation. Since D is finite it is equivalent to define the permutations as one to one functions, or as reversable functions. The set of all permutations of D is denoted by S, hence $S \in D^D$. More formally S is defined by: $S=\{f \mid f \in D^D, \{f(x) \mid x \in D\}=D\}$.
If $f, g \in S$ then the function h defined by $h(x)=g(f(x))$ also belongs to S. The permutation h is called the product of the permutations f and g and is denoted by gf. The set S is a group with respect to that product. It is called the symmetric group. The identity element will be denoted by e, hence $e(x)=x$ for all $x \in D$. The group is not abelian as can be seen from the example:

x	0	1	2	3	4	5	6	7	8	9
f(x)	1	0	2	3	4	5	6	7	8	9
g(x)	0	2	1	3	4	5	6	7	8	9
fg(x)	1	2	0	3	4	5	6	7	8	9
gf(x)	2	0	1	3	4	5	6	7	8	9

The inverse of permutation f is denoted by f^{-1} and hence $ff^{-1}=f^{-1}f=e$.

Theorem 3.0 If $f_i \in S$ for all i, then the code C is E_1-proof.
Proof: Two words differing only on the j-th position cannot both belong to C, since otherwise

$$\sum_{i=1}^{j-1} f_i(a_i)+f_j(a_j)+\sum_{i=j+1}^{n} f_i(a_i)=\sum_{i=1}^{j-1} f_i(a_i)+f_j(a_j')+\sum_{i=j+1}^{n} f_i(a_i)$$

would hold. By cancelling equal terms from the left and the right, $f_j(a_j)=f_j(a_j')$ would follow, so that from $f_i \in S$ the contradiction $a_i=a_i'$ can be derived.

The converse of this theorem is not true since a code C may be E_1-proof while not all the functions f_i belong to S. Counter example: Let the functions f_1, f_2 and f_3 be defined by the table:

x	0	1	2	3	4	5	6	7	8	9
$f_1(x)$	2	4	6	0	1	1	1	1	1	1
$f_2(x)$	4	6	0	2	1	1	1	1	1	1
$f_3(x)$	6	0	2	4	1	1	1	1	1	1

The words $a_1a_2a_3$ satisfying $f_1(a_1)+f_2(a_2)+f_3(a_3)=0$ obviously cannot contain any digit "higher" than 3 and therefore the code is E_1-proof since from $f_i(a_i)=f_i(a_i')$ it follows again that $a_i=a_i'$, for $a_i, a_i' \leq 3$.

In the rest of this chapter it will be assumed that all f_i's are permutations. It will also be assumed in this chapter that the group (D,+) is abelian. It is well-known from the theory of groups that this group has to be the cyclic group of order 10. It is also well-known that the additive group modulo 10 is cyclic. Those readers not familiar with group theory may therefore interpret the operation + as addition modulo 10. In agreement with this interpretation the elements of D will be denoted by the decimals, or $D=\{0,1,2,3,4,5,6,7,8,9\}$.

3.1. Formulation of the requirements.

The condition that a code C as defined above is transposition-proof is that $f_i(a_i)+f_{i+1}(a_{i+1})\neq f_i(a_{i+1})+f_{i+1}(a_i)$ for all a_i, a_{i+1} with $a_i \neq a_{i+1}$. Now define x and y by $x=f_i(a_i)$ and $y=f_i(a_{i+1})$, then $a_i=f_i^{-1}(x)$ and $a_{i+1}=f_i^{-1}(y)$ and the condition becomes $x+f_{i+1}f_i^{-1}(y)\neq y+f_{i+1}f_i^{-1}(x)$, or since the addition is abelian, $x-f_{i+1}f_i^{-1}(x)\neq y-f_{i+1}f_i^{-1}(y)$ follows. The condition has to hold for all x and y with $x \neq y$. In other words the function g defined by $g(x) = x - f_{i+1}f_i^{-1}(x)$ has to be a permutation. Loosely said $x - f_{i+1}f_i^{-1}(x)$ has to be a permutation or more formally

$\{x-f_{i+1}f_i^{-1}(x) | x \in D\} = D$. In an analogous way it can be deduced that the twin errors are detected if $x+f_{i+1}f_i^{-1}(x)$ is a permutation.
Jump transposition detection requires that
$f_i(a_i)+f_{i+1}(a_{i+1})+f_{i+2}(a_{i+2}) \neq f_i(a_{i+2})+f_{i+1}(a_{i+1})+f_{i+2}(a_i)$.
Setting $x=f_i(a_i)$ and $y=f_i(a_{i+2})$ and cancelling the middle terms gives $x+f_{i+2}f_i^{-1}(y) \neq y+f_{i+2}f_i^{-1}(x)$ so that now $x-f_{i+2}f_i^{-1}(x)$ has to be a permutation. Again in an analogous way the condition for the detection of the jump twin errors can be reduced to the requirement that $x+f_{i+2}f_i^{-1}(x)$ has to be a permutation.
Finally the phonetic errors are detected if
$f_i(a)+f_{i+1}(0) \neq f_i(1)+f_{i+1}(a)$ holds for $a \neq 0,1$. This inequality is however also valid for 0 and 1, so that, after setting $x=f_i(a)$, it follows that $x-f_{i+1}f_i^{-1}(x) \neq f_i(1)-f_{i+1}(0)$ has to be true for all $x \in D$.
In the list below the conditions are summarized.

Transpositions	$\{x-f_{i+1}f_i^{-1}(x) \mid x \in D\} = D$
Twin errors	$\{x+f_{i+1}f_i^{-1}(x) \mid x \in D\} = D$
Jump transpositions	$\{x-f_{i+2}f_i^{-1}(x) \mid x \in D\} = D$
Jump twin errors	$\{x+f_{i+2}f_i^{-1}(x) \mid x \in D\} = D$
Phonetic errors	$f_i(1)-f_{i+1}(0) \notin \{x-f_{i+1}f_i^{-1}(x) \mid x \in D\}$.

3.2. Analysis of the conditions.

The five requirements are not compatible since the first one contradicts the last one. Suppose that $\{x-f_{i+1}f_i^{-1}(x) \mid x \in D\} = D$, then it follows, as $f_i(1)-f_{i+1}(0) \in D$, that the last condition is not fulfilled.
Fortunately or rather unfortunately no one of the first four conditions can be satisfied because of the next theorem, so that it becomes theoretically possible to satisfy the fifth one.
Theorem 3.2.0 There does not exist a permutation f of the 2k residue classes 0,1,...,2k-1 such that the function g defined by $g(x)=f(x)+x$ is also a permutation.
Proof: Consider $\sum(f(x)+x)=\sum f(x)+\sum x=2\sum x=2k(2k-1)=0 (\text{mod } 2k)$, but if $f(x)+x$ were a permutation then $\sum(f(x)+x)=\sum x=k \text{ mod } 2k)$ would hold.

As a corollary it follows that, if in $\{f(x)+x \mid x=0,\dots,2k-1\}$ just one element is missing, say i, then i+k (mod 2k) has to appear twice.
By applying this theorem on the permutations $\pm f_{i+1}f_i^{-1}$ and $\pm f_{i+2}f_i^{-1}$ with k=5 it follows that the first four conditions are impossible. This is the result referred to in 2.1, which led to the belief that no pure decimal check could yield a E_1^n- and transposition-proof code.
The question remains of how close the ideal can be approximated by using addition modulo 10. The next best to being a permutation is that the set $\{ x-f(x) \mid x \in D \}$ lacks only one element. The expression "f(x)-x is nearly a permutation" will be used in the sequel if such is the case.
The I.B.M. code of 2.3 is an example:

x	0	1	2	3	4	5	6	7	8	9
f(x)	0	2	4	6	8	1	3	5	7	9
x-f(x)	0	9	8	7	6	4	3	2	1	0

Note that the digit 5 is missing and that the 0 occurs twice.
The set of permutations f such that x-f(x) is nearly a permutation is denoted by P. The subset of P consisting of those permutations f for which x+f(x) is also nearly a permutation will be denoted by Q, hence $Q \subseteq P \subseteq S$.
If $g_i \in Q$ and if $f_{i+1}=g_i f_i$ for $1 \leq i < n$ and if f_1 is arbitrarily chosen in S, then the code defined by $\sum_{i=1}^{n} f_i(a_i)=c$ is "as good as possible" in detecting the transpositions and the twin errors. The converse is also true.

In view of the large number of permutations ($|S|$ =3628800=10!) a computer program has been written to find the sets P and Q. Simply generating all permutations and rejecting the ones for which x-f(x) is not nearly a permutation, is not only inefficient but also unimaginative. It is much nicer to generate the permutations lexicographically and to test while each permutation is built up. Building each permutation by first choosing f(0), then f(1) etc. is a multiple stage decision process and the idea of dynamic programming can be applied. If for instance f(0),f(1) and f(2) are chosen so that 0-f(0)=1-f(1)=2-f(2), then all 7! further codes may be skipped. This would also be the case, according to the corollary

of theorem 3.2.0, if $0-f(0)=1-f(1)=2-f(2)+5$. As will be seen below, a further saving, by at least a factor 150, is gained through a study of the structure of the sets P and Q. There are four transformations which leave both P and Q invariant, and which define equivalence relations in these sets.

3.3. Detection-rate preserving transformations.

The permutations h_a defined by $h_a(x)=x+a$, with $x,a \in D$, form a representation of (D,+) in the symmetric group S, since $h_a\, h_b(x))=h_a(x+b)=x+b+a=x+a+b=h_{a+b}(x)$. The set H defined by $H=\{h_a \mid a \in D\}$ is a subgroup of S isomorphic with (D,+).

If $f \in P$ (or Q), then the double cosets $HfH(=\{h_a f h_b \mid a,b \in D\})$ belong to P (or Q), since $x' \pm h_a f h_b(x') = y' \pm h_a f h_b(y')$, with $x'=x+b$ and $y'=y+b$, follows immediately from $x \pm f(x) = y \pm f(y)$. Hence the transformations $f \rightarrow h_a f$, with $a \in D$, leave both P and Q invariant. The sets $Hf(= \{hf \mid h \in H\}$) obviously contain 10 different permutations, among which there is just one which has 0 as a fixed point; i.e. $h_{-f(0)}f$, as $h_{-f(0)}f(0)=f(0)-f(0)=0$

The search may therefore be limited to the sets P_0 and Q_0 defined by $P_0= \{f \mid f \in P, f(0)=0\}$ and $Q_0= \{f \mid f \in Q, f(0)=0\}$. Clearly $|P_0| = |P| /10$ and $|Q_0| = |Q| /10$. By setting $f(0)=0$ there are 9! possibilities left and the search is cut down by a factor 10.

The transformations G_a defined by $G_a f=h_{-f(a)} f h_a$ leave P_0 and Q_0 invariant, since $\{h_{-f(a)} f h_a \mid a \in D\} \subset HfH$ and since $G_a f(0)=f(a+0)-f(a)=0$. That the sets $\{G_a f \mid a \in D\}$, for $f \in P_0$, contain 10 permutations each is true, but not trivial. It hinges on the circumstance that $x-f(x)$ is nearly a permutation for all $f \in P_0$. For such a func ion there are two elements d_1 and d_2 such that $d_1-f(d_1)=d_2-f(d_2)$. Let these elements be called the duplicators. The duplicators of $G_a(f)$ are d_1-a and d_2-a, since $f(d_1-a+a)-f(a)-(d_1-a)=f(d_1)-d_1+a-f(a)=$

$=f(d_2)-d_2+a-f(a)=f(d_2-a+a)-f(a)-(d_2-a)$. If $d_1-d_2 \neq 5$ then the ten values of a , give 10 different permutations, as they have different duplicators. But if $d_1-d_2=5$ then it is conceivable that $f(x)=f(x+5)-f(5)$ since the functions on both sides of the equation have the same duplicators. However if this were the case, then $f(5)= (5+5)-f(5)=- (5)$, or $2f(5)=0$. But since $f(0)=0$, it follows that $f(5)=5$ so 0 and 5 are both

fixed points of f and hence duplicators. Substituting 1 for x gives $f(1)=f(1+5)-5$ or $f(1)-1=f(6)-6$, which makes 1 and 6 also duplicators and consequently $x-f(x)$ is not nearly a permutation. Now that it has been established that each set has to have 10 members the problem remains to select appropriate representatives in order to facilitate the search. The effect of the transformation G_a is that the duplicators are both shifted modulo 10. Their cyclic distance is therefore an invariant. Let P_{0i} (resp Q_{0i}) be the subset of P_0(resp Q_0) consisting of the permutations f such that 0 and i are the duplicators of $x-f(x)$. Now each permutation $f \in P_0-P_{05}$ has just one equivalent permutation in one of the classes $P_{01}, P_{02}, P_{03}, P_{04}$. The permutations of P_{05} will have two equivalent permutations in P_{05}. It is therefore only necessary to find the sets P_{0i}, for $i<6$, and $|P_0| = 10 \sum_{i=1}^{4} |P_{0i}| +5 |P_{05}|$. It is much easier to search for the permutations of P_{0i}, since not only two values are fixed, but also because the test is simpler now, as no more duplications in $x-f(x)$ are allowed. Moreover according to the corollary of theorem 3.2.0 $f(x)\neq x+5$ has to hold.

The search can further be limited by yet another transformation-type, which leaves each P_{0i} (and Q_{0i}) invariant. This transformation is based on the automorphisms of the cyclic group C_{10}. An automorphism is a permutation δ of the elements of the group satisfying : $\delta(a+b)= \delta(a)+ \delta(b)$. The automorphisms form a subgroup of the symmetric group. In the case of C_{10} it is a cyclic group with 4 elements. The formula above suggests a multiplication and indeed multiplication modulo 10, by a factor relatively prime with 10, does the trick. These factors are 1,3,7,9. It should be noted that if $(a,10)\neq 1$ then ax (mod 10) is not even nearly a permutation of x. The automorphisms are generated by 3 (or 7), since $1=3^0$, $3=3^1$, $7=3^3$ and $9=3^2$. The order of an element a is defined as the lowest positive integer k, such that $\sum_{i=1}^{k} a=0$. It is well-known from the theory of groups that the automorphisms leave the order invariant. The group C_{10} has 4 elements of the order 10 i.e. 3,9,7,1 and 4 elements of the order 5, i.e. 2,4,6,8. 0 is the only element of the order 1 and 5 has the order 2. Hence $\delta(0)=0$ and $\delta(5)=5$ for each automorphism δ .To each automorphism δ there corresponds a transform F_δ defined by $F_\delta(f)= \delta f \delta^{-1}$, it is known as the transform of

f by δ. If f is given by a table $f=\left\{ \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ i_0 & i_1 & i_2 & i_3 & i_4 & i_5 & i_6 & i_7 & i_8 & i_9 \end{matrix} \right\}$

then $F_\delta(f)$ is obtained by applying δ to all the symbols of both entries of the table. (see 8).

Suppose that $f \in P_{0i}$ and that $\delta i=j$ and let $g=F_\delta(f)$, then $g(j)=\delta f \delta^{-1}(\delta i)=\delta f(i)=\delta(i)=j$ and $g(0)=\delta f \delta^{-1}(0)=\delta f(0)=\delta(0)=0$. Moreover, since x-f(x) is nearly a permutation, it follows that x-g(x) is so too, by remarking that $\{x-g(x)\} = \{\delta\delta^{-1}(x)-\delta f \delta^{-1}(x)\} = \{\delta(\delta^{-1}(x)-f(\delta^{-1}(x)))\} = \delta\{y-f(y)\}$. Hence $g \in P_{0j}$. Now it is possible to project P_{04} on P_{02} and P_{03} on P_{01} by the transforms F_3 and F_7. The class P_{05} however is left invariant and though this third transformation induces an equivalence in P_{05} it is hard to capitalize upon this fact. It is also tempting to try to split P_{0i} further by using the transformation g(x)=-f(-x), which projects P_{0i} on P_{0-i} and to go back to P_{0i} by the transformation of the second type h(x)=g(x-i)-g(-i). The resulting transformation is h(x)=i-f(i-x). This transformation does not necessarily lead to a different permutation, as can be seen in the first table below for i=1. In P_{05} the transform F_9 has also its invariants. An example is given in the second table below. Note that x+f(x) is also nearly a permutation in this second example.

x	0	1	2	3	4	5	6	7	8	9
f(x)	0	1	3	5	7	9	2	4	6	8
x-f(x)	0	0	9	8	7	6	4	3	2	1
1-x	1	0	9	8	7	6	5	4	3	2
f(1-x)	1	0	8	6	4	2	9	7	5	3
1-f(1-x)	0	1	3	5	7	9	2	4	6	8
x+f(x)	0	2	5	8	1	4	8	1	4	7

x	0	1	2	3	4	5	6	7	8	9
f(x)	0	2	6	1	7	5	3	9	4	8
x-f(x)	0	9	6	2	7	0	3	8	4	1
-x	0	9	8	7	6	5	4	3	2	1
f(-x)	0	8	4	9	3	5	7	1	6	2
-f(-x)	0	2	6	1	7	5	3	9	4	8
x+f(x)	0	3	8	4	1	0	9	6	2	7

It may be worthwhile to note that there is yet a fourth transformation I, which leaves P (and Q) invariant. It is defined by $I(f)=f^{-1}$. The invariance follows by substituting $f^{-1}(y)$ for x in the relation $|\{x-f(x)\}| = 9$, for $\{f(y)-y\}$ will have the same number of elements. Since moreover the inverse of a permutation f has the same fixed points as f, also P_{0i} (and Q_{0i}) is left invariant. It can happen that I(f)=f, as is shown by the example on page 69.

x	0	1	2	3	4	5	6	7	8	9
f(x)	0	1	3	2	7	9	8	4	6	5
x-f(x)	0	0	9	1	7	6	8	3	2	4
$f^{-1}(x)$	0	1	3	2	7	9	8	4	6	5
x+f(x)	0	2	5	5	1	4	4	1	4	4

Even though some of the transformations mentioned in this section are not used to relieve the search, they still are helpful for checking the output.

The result of this section is that it is only necessary to find the sets P_{01}, P_{02} and P_{05} (or Q_{01}, Q_{02}, Q_{05}). The set P (or Q) can be found afterwards by applying certain transformations on these sets. The following equalities hold: $|P| = 200(|P_{01}| + |P_{02}|) + 50 |P_{05}|$ and $|Q| = 200(|Q_{01}| + |Q_{02}|) + 50 |Q_{05}|$.

3.4. <u>The search program</u>.

In the program the permutations are built by means of a multiple decision process. In 8 stages the process is ready, since f(0)=0 and f(i)=i has to hold if $f \in P_{0i}$. The available function values are stored as a chain in an array called chain (-1:9). Each time that a digit j is allocated to some f(i) the chain is shortcircuited by the assignment chain (k):=chain (j). k is supposed to be the previous value which was possible, for 0 the dummy value -1 is taken as the previous one. So chain(-1) refers to the first digit which is still available, chain(chain(-1)) to the second one and so on.

The crucial part of the program is the test for the feasibility of an allocation. In a boolean array called difference (0:9) the occurrence of a difference i-f(i) is memorized by making difference (i-f(i)) true as some value has been allocated to f(i). The fields difference (0) and difference (5) are initialized by true, since 0 and 5 are not allowed as a difference i-f(i). In order to see whether j is possible as value for f(i) the program simply tests whether difference (i-j) is false or not.

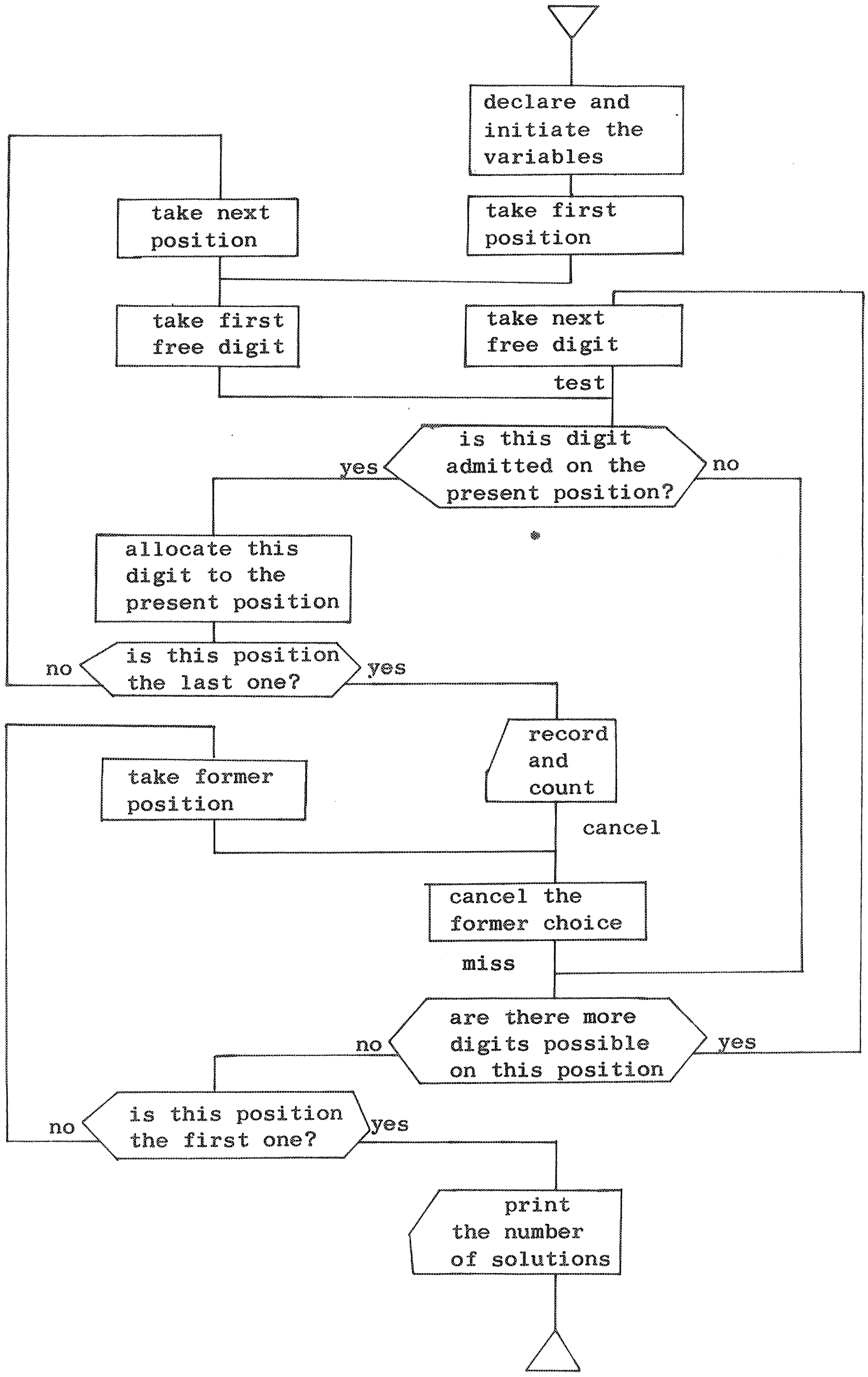
In the array f(0:9) the allocated value is stored and in the array choice (0:9) the value previous to the one allocated is memorized.

The latter array is necessary to be able to revoke a decision, if the process comes to a stop at a higher level. After revoking a decision the feasibility of the next possible value is tested. If there is no next value available, the decision at the previous stage has to be revoked, and so on.The process also stops temporarily when a permutation satisfying the conditions has been found. It is then counted in the array count (1:5) and it can be tested whether the permutation also belongs to Q, if so, a print-out is requested. After that, the process is started again by revoking the last decision.

The flowchart of the program is given on page 71, for the benefit of those readers who prefer this less clear but more general description over the precise list of instructions. The latter is given all the same to make the details available.

The structure of the program, as given in the flowchart is quite general. If for instance the test is skipped, the program will generate all permutations in lexicographical order. If however $f(i) \neq i$ is used as test condition, the non-concurrent permutations are generated. The same flowchart can be used for more complicated problems, like the pentomino-fitting-puzzles (see 15).

In chapter 5 it will be used again.


declare and initiate the variables
take first position
take next position
take first free digit
take next free digit
test
is this digit admitted on the present position?
yes
no
allocate this digit to the present position
is this position the last one?
no
yes
record and count
take former position
cancel
cancel the former choice
miss
are there more digits possible on this position
no
yes
is this position the first one?
no
yes
print the number of solutions

```
'begin'             'integer' d,j,k,x,y;
                    'integer' 'array' f,choice(0:9),chain(-1:9),
                    count(1:5);
                    'boolean' 'array' difference(0:9);
                    'for' j:=1,2,5 'do'
  'begin'           'for' x:=0 'step' 1 'until' 9 'do'
    'begin'         chain(x):=x+1; difference(x):='false'
    'end';          count(j):=0; f(j):=j;
                    difference (0):=difference(5):='true';
                x:=chain(-1):='if' j 'equal' 1 'then' 2 'else' 1;
                    'if' j 'greater' 1 'then' chain(j-1):=j+1;
start;                      k:=-1;
test:               y:=chain(k);d:=x-y; 'if' d 'less'  0 'then' d:=d+10;
                    'if' difference(d) 'then' 'go to' miss;
allocate:           f(x):=y; chain(k):=chain(y); choice(x):=k;
                    difference(d):='true'; 'if' x 'less' 9 'then'
    'begin'         x:=x+1;'if' x 'equal' j 'then' x:=x+1;
                    'go to' start
    'end';          count(j):=count(j)+1;
discard:            x:=9;
cancel:             k:=choice(x); y:=f(x); chain(k):=y; d:=x-y;
                    'if' d 'less' 0 'then' d:=d+10;difference(d):='false';
miss:               'if' chain(y) 'less' 10 'then'
    'begin'         k:=y;'go to' test
    'end';          'if' x 'greater' i 'then'
      'begin'       x:=x-1; 'if' x 'equal' j 'then'
         'begin'    'if' j 'equal' 1 'then''go to' ready; x:=x-1
         'end';     'go to' cancel
     'end';
ready:              nlcr(1); write("the number of permutations in p0");
                    type(j); write(" is "); type(count(j));
  'end';            d:=50*(4*(count(1)+count(2))+count(5));
                    write("the total number of permutations in p is");
                    type(d)
 'end'
```

As a result of the program it turns out that $|P_{01}| = |P_{02}| = 104$ and that $|P_{05}| = 96$. Hence $|P| = 200(104+104)+50\times96=46400$.
Furthermore $|Q_{01}| = 2$; $|Q_{02}| = 8$ and $|Q_{05}| = 16$, so that $|Q| = 200(2+8)+50\times16=2800$.

Since Q is not empty it is natural to disregard the rest of P, at least temporarily.

The permutations of Q_{01} are:

x	0	1	2	3	4	5	6	7	8	9
$q_1(x)$	0	1	4	6	3	9	2	8	5	7
$q_2(x)$	0	1	6	4	2	8	3	9	7	5

The transformation $f(x)\to1-f(1-x)$ leaves both permutations invariant, but the transformation $f(x)\to f^{-1}(x)$ interchanges them.

The permutations of Q_{02} are:

x	0	1	2	3	4	5	6	7	8	9
$q_3(x)$	0	4	2	5	8	1	3	6	9	7
$q_4(x)$	0	4	2	9	3	6	8	1	5	7
$q_5(x)$	0	5	2	6	1	3	7	9	4	8
$q_6(x)$	0	5	2	9	6	3	7	4	1	8
$q_7(x)$	0	7	2	4	1	8	5	9	6	3
$q_8(x)$	0	7	2	4	8	3	5	9	1	6
$q_9(x)$	0	8	2	5	3	6	9	1	4	7
$q_{10}(x)$	0	8	2	5	7	1	4	6	9	3

In Q_{02} the transformation $f(x)\to2-f(2-x)$ interchanges q_3 and q_9; q_4 and q_{10}; q_5 and q_8; q_6 and q_7, whereas the pairs q_3 and q_5; q_4 and q_7; q_6 and q_{10}; q_8 and q_9 are each others inverse.
In Q_{05} the transformations $f(x)\to f(x+5)+5$, $I(f)=f^{-1}$ and $F_3(f(x))=3f(7x)$, $F_9(f(x))=-f(-x)$ and $F_7(f(x))=7f(3x)$ are of interest.
Q_{05} has 16 permutations, which are tabulated on the next page.

The transformation $f(x)\to f(x+5)+5$ interchanges the pairs: (q_{11},q_{26}); (q_{12},q_{25});(q_{13},q_{18});(q_{14},q_{16});(q_{15},q_{22});(q_{17},q_{20});(q_{19},q_{24});(q_{21},q_{23}).
The group of transformations F_{3^i} with $1 \leq i < 4$, splits Q_{05} into 6 classes i.e. 4 classes with 2 elements and 2 classes with each 4 elements. In each class the elements are interchanged cyclicly.

x	0	1	2	3	4	5	6	7	8	9
$q_{11}(x)$	0	2	4	9	7	5	3	1	6	8
$q_{12}(x)$	0	2	6	1	7	5	3	9	4	8
$q_{13}(x)$	0	2	6	9	3	5	8	4	1	7
$q_{14}(x)$	0	2	9	6	3	5	8	1	4	7
$q_{15}(x)$	0	3	1	6	8	5	2	4	9	7
$q_{16}(x)$	0	3	6	9	2	5	7	4	1	8
$q_{17}(x)$	0	3	9	4	8	5	2	6	1	7
$q_{18}(x)$	0	3	9	6	2	5	7	1	4	8
$q_{19}(x)$	0	7	1	4	8	5	3	9	6	2
$q_{20}(x)$	0	7	1	6	2	5	8	4	9	3
$q_{21}(x)$	0	7	4	1	8	5	3	6	9	2
$q_{22}(x)$	0	7	9	4	2	5	8	6	1	3
$q_{23}(x)$	0	8	1	4	7	5	2	9	6	3
$q_{24}(x)$	0	8	4	1	7	5	2	6	9	3
$q_{25}(x)$	0	8	4	9	3	5	7	1	6	2
$q_{26}(x)$	0	8	6	1	3	5	7	9	4	2

The classes are: $(q_{11},q_{15});(q_{12},q_{20});(q_{17},q_{25});(q_{22},q_{26});(q_{19},q_{23},q_{21},q_{24});(q_{13},q_{16},q_{18},q_{14})$.
The transformation I splits Q_{05} into 4 classes of 2 members each which are each others inverse. The pairs are: $(q_{11},q_{20});(q_{12},q_{15});(q_{13},q_{23});(q_{14},q_{19});(q_{16},q_{24});(q_{17},q_{26});(q_{18},q_{21});(q_{22},q_{25})$.
Q_{05} is generated by means of the transformations mentioned above from the permutations q_{11} and q_{13}.
Define Q'_{05} by $Q'_{05} = \{q_{11},q_{12},q_{13},q_{15},q_{16},q_{19},q_{21},q_{23}\}$.
The set Q'_{05} is chosen in such a way that the transformation $f(x)\to f(x+5)+5$ has no equivalent permutations in Q'_{05}. Therefore, if $Q'=Q_{01}\cup Q_{02}\cup Q_{03}\cup \cup Q_{04}\cup Q'_{05}$ it follows that the set Q' together with the 100 transformations $f(x)\to f(x+a)+b$ generate the set Q. In the next section it will be seen that this concise description of the set Q, as a by-product of the hunt for shortcuts in the search, is a great asset in itself. Now that the(first) search is over it is still useful to try to get the set Q in a tighter grip by means of the transformation group F_{3^i}. This group splits the set $Q_{02}\cup Q_{04}$ into classes of 4 elements each. The set $\{q_3, q_4, q_5, q_7\}$ denoted by Q'_{02} has a representative from each class. The set $Q_{01}\cup Q_{03}\cup Q'_{05}$ however is split into classes with 2 elements each, if $Q''_{05} =\{q_{11},q_{13},q_{20},q_{23}\}$ then $Q_{01}\cup Q''_{05}$ has a representative of each class, so that Q is known as soon as $Q''=Q_{01}\cup Q'_{02}\cup Q''_{05}$ is known. By using the inverse of the permutations a still greater reduction can be achieved, in fact Q is generated by the set $\{q_1,q_3,q_4,q_{11},q_{13}\}$.

3.5. The detection of the jump errors.

The condition for the detection of the jump transpositions was that the function $x-f_{i+2}f_i^{-1}(x)$ is nearly a permutation. In the foregoing section it was shown that the code detects the transpositions and the twin errors optimally if $g_i=f_{i+1}f_i^{-1}\in Q$. It is therefore advisable to apply permutations f_i such that $g_i=f_{i+1}f_i^{-1}\in Q$ and that the consecutive g_i's satisfy the condition: that $x-g_{i+1}g_i(x)$ is nearly a permutation, or that at least the equality $x-g_{i+1}g_i(x)=y-g_{i+1}g_i(y)$ is valid for the minimum number of pairs x,y. If the first is the case the permu-

tations g_{i+1} and g_i are said to be matching. A chain of permutations $g_i \in Q$ is wanted such that the adjacent g's form matching pairs. An obvious strategy for finding such chains is to take a $g_1 \in Q$ at random and to search for a $g_2 \in Q$ matching with g_1, and so on. This is, especially if the matching pairs are scarce, very inefficient. It is better to make a catalogue of the matching pairs first and to build the chains with the aid of that catalogue afterwards. The matching pairs have to be selected out of the 7840000 pairs from QxQ. It seems worthwhile to investigate whether the representation of Q by means of the set Q' and the transformations $f(x) \to f(x+a)+b$ yields a saving in labour. Let $f, f' \in Q$, then for some g and g' with $g, g' \in Q'$; $f(x)=g(x+a)+b$ and $f'(x)=g'(x+a')+b'$ hold. The pair f,f' matches if $x-g(g'(x+a')+b'+a)-b$ is nearly a permutation, which is equivalent with the requirement that $y-g(g'(y)+c)$ with $b'+a=c$, is nearly a permutation. So if $g(x+c)$ and $g'(x)$ match, with $g, g' \in Q'$ and $c \in D$, then $g(x+a)+b$ and $g'(x+a')+b'$, with $b'+a=c$, also match. This remark reduces the number of pairs, which have to be tested, by a factor 1000.

A further reduction is obtained with the aid of the transformation F_3, since it may be assumed that the second member of the pair lies in Q". If the second member lies outside Q" then the transformation F_3 will bring it into Q" and from

$x-g(g'(x)+c)=7\cdot 3x-7\cdot 3g(7(3g'(7\cdot 3x)+3c)=7(y-F_3g(F_3g'(y)+3c))$ it follows that F_3g, F_3g' match if g,g' do. Hence only the 2800 triplets g,g',c with $g \in Q'$; $g' \in Q''$ and $c \in D$ have to be tested. A simple program gives as sad result that no matching pairs exist. This is in agreement with the fact that up to now no code was known (at least to the author) which had the property to detect both the transpositions and the jump transpositions nearly optimal. In trying to prove that these codes do not exist the following example was found:

x	0	1	2	3	4	5	6	7	8	9
f'(x)	0	9	6	4	1	5	8	3	7	2
f(x)	0	7	3	6	1	5	4	9	2	8
ff'(x)	0	8	4	1	7	5	2	6	9	3
x-f'(x)	0	2	6	9	3	0	8	4	1	7
x-f(x)	0	4	9	7	3	0	2	8	6	1
x-ff' (x)	0	3	8	2	7	0	4	1	9	6
x+f'(x)	0	0	8	7	5	0	4	0	5	1
x+f(x)	0	8	5	9	5	0	0	6	0	7
x+ff'(x)	0	9	6	4	1	0	8	3	7	2

Both permutations leave 7 (i.e. $\binom{4}{2}+\binom{2}{2}$) twin errors undetected, but the jump twin errors are detected nearly optimal.

There are two possible approaches now:

i) Admitt also permutations which do not optimalize the detection of the twin errors or;

ii)suboptimalize the detection of the jump transpositions. By means of a simple computer program it appears that the permutations of P_{01}, P_{02} and P_{05} are divided with respect to their twin error detection capacity according to the following table:

undetected twin errors	1	2	3	4	5	6	7	8	9	10	15	16
number of permutations	26	0	40	62	54	22	48	16	4	26	4	2

Following the first suggestion would thus imply the loss of two more twin errors, as the second class is empty. The second suggestion should therefore be preferred, if it means the loss of only one more jump transposition. Such is indeed the case as will be seen in the sequel. Again the search may limited to the triplets g,g',c, with $g \in Q'$;$g' \in Q''$ and $c \in D$. A pair g,g' is said to be nearly matching if $x-gg'(x)=y-gg'(y)$ holds for only two pairs x,y.

By a modified program all the triplets are now tested to see whether there are nearly matching pairs. For the approved triplets it is counted how many pairs x,y satisfy $x+g(g'(x)+c)=y+g(g'(y)+c)$. It turns out that 32 pairs are the topscorers, each leaving two jump transpositions and

two jump twin errors undetected. Eight permutations are involved in these 32 pairs. These permutations fall apart into two families. The members of one family are the inverse of those of the other family. Within each family any two can form a nearly matching pair with an appropriate value for c. The permutations of one of the families are denoted by h_1, h_2, h_3, h_4 and $h_i(x+a)+b$ nearly matches $h_j(x+a')+b'$ provided that $b'+a=c_{ij}$, where c_{ij} is listed below.

x	0	1	2	3	4	5	6	7	8	9
$h_1(x)$	0	4	2	5	8	1	3	6	9	7
$h_2(x)$	0	8	2	5	3	6	9	1	4	7
$h_3(x)$	0	2	9	6	4	1	8	5	7	3
$h_4(x)$	0	8	5	2	4	1	7	9	6	3

c_{ij}	1	2	3	4
1	3	5	0	6
2	5	7	2	8
3	2	4	9	5
4	8	0	5	1

It is easy now to produce chains of any length, consisting of nearly matching permutations. As a matter of fact $h_i(x+c_{ii})$ provides an infinite chain with equal links, resulting in a (periodic) progressive code. Codes based on these chains detect 97.8% of the transpositions, 97.8% of the twin errors, 95.6% of the jump transpositions and 95.6% of the jump twin errors, The problem of selecting chains which detect the phonetic errors optimally is dealt with in the next section.

3.6. The detection of the phonetic errors.

A phonetic error on the positions i and i+1 is detected if
$x-f_{i+1}f_i^{-1}(x)\neq f_i(1)-f_{i+1}(0)$ for $x\in D$, or
$x-g_i(x)\neq f_i(1)-g_i(f_i(0))$ with $f_{i+1}=g_i f_i$.

The righthand side is a fixed value, which has to correspond with the missing value of the set $\{x-g_i(x)\mid x\in D\}$. The Colenbrander codes (see section 2.3) defined by $\sum f^i(g(a_i))=0 \pmod{10}$, with

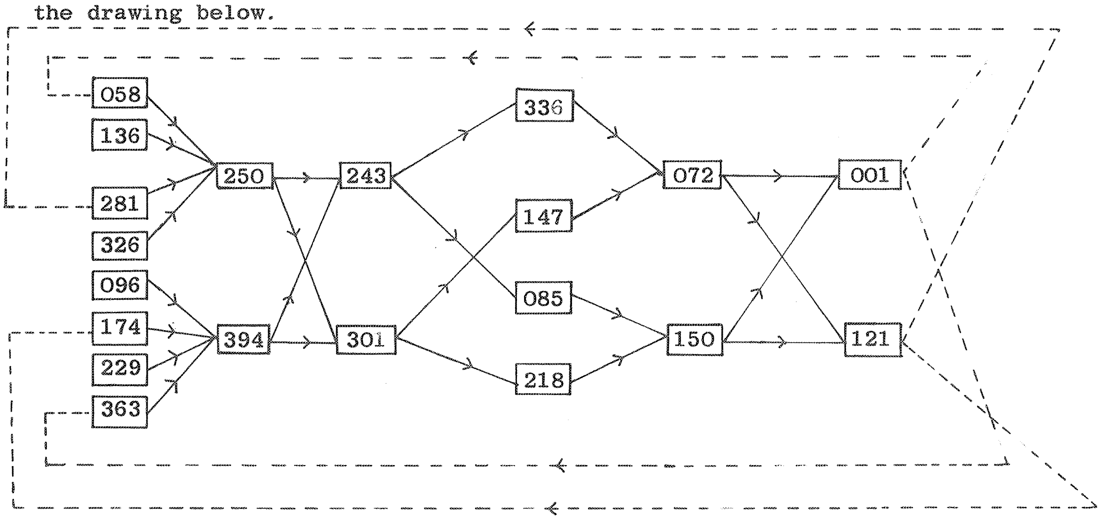
$f=\binom{0123456789}{9246801357}$ and where g is an arbitrary permutation, admitt an elegant solution.

In $\{x-f(x)\mid x\in D\}$ the value 0 is missing and by choosing g so that $f(g(0))=g(1)$ it is achieved that $x+f^{i+1}(g(0))\neq f^i(g(1))+f(x)$ is

equivalent with $x \neq f(x)$, which is valid for all x since the zero was missing. The choice $g = \{{0123456789 \atop 9736124850}\}$, which is the one used in the tests of section 2.4, is a proper one. The general case is more difficult. For the functions h_i given in the preceding section, this missing value is 5, and for $h_i(x+a)+b$ it is 5-a-b. Suppose that a chain $g_1, g_2, \ldots, g_k$ with the property that the code based on the g's in the usual manner, detects all phonetic errors. Suppose furthermore that the g's are selected from $\{h_1, h_2, h_3, h_4\}$ and that the selected h_i's are suitably transformed as set forth in the section above. The question arises as to the conditions for prolongation of the chain. Let $f_k(0)=q$ and $f_k(1)=p$ and let g_k be a transform of h_s. If the next link of the chain is a transform of h_t, then it has to be $h_t(x+c_{ts})$. The missing value of $\{x-h_t(x+c_{ts}) \mid x \in D\}$ is $5-c_{ts}$ and hence the condition for detecting the phonetic errors is $p-h_t(q+c_{ts})=5-c_{ts}$. This implies that for each t only certain triplets for p,q,s, with $p,q \in D$ and $s \in \{1,2,3,4\}$, admitt the prolongation of the chain by the permutation $h_t(x+c_{ts})$. After the prolongation a new triple $h_t(p+c_{ts})$, $h_t(q+c_{ts})$, t arises, which does or does not admitt further extension of the chain. For each choice of t and s there are 10 pairs p,q which satisfy the condition $p-h_t(q+c_{ts})=5-c_{ts}$. Hence there are altogether 160 triplets, namely $5-c_{ts}+h_t(q+c_{ts})$, q, s which are suitable for chain extension. The resulting triplets after the extension are $h_t(5+h_t(q+c_{ts})), h_t(q+c_{ts})$, t, with $q \in D$ and $s, t \in \{1,2,3,4\}$. In terms of the theory of graphs the problem is to find, possibly circuits, but at least the maximal paths in the directed graph defined by the coupling of the triplets. The triplets itself are the points of the graph. Let these points be denoted by T_i with $0 \leq i \leq 399$. The edges of the graph are the 160 ordered pairs (T_i, T_j). The vertex T_i is called the initial vertex and T_j is called the terminal vertex of the edge. An edge (T_i, T_j) with a terminal vertex, which is not the initial vertex of any other edge, is called a twig.

Obviously a twig cannot be part of a circuit and removing the twigs from the graph will not eliminate any circuits. The length of the maximal paths however will be deminished by one. The following procedure is applied in order to find the maximal paths:

i) Find the twigs of the graph. This can be done efficiently by having the list of the edges ordered according to the initial vertex.
ii) Remove the twigs from the edge-list and put them on the first pruning-list.
iii) Repeat the pruning until there are no more twigs left. If the graph is pruned away after n prunings then the length of the longest chains is also n, otherwise there has to be a circuit. The longest chain can be reconstructed from the pruning-lists, starting from the back.
In the present case it turns out that there are 64 chains having the maximal length of 6. These chains are intertwined in a manner shown in the drawing below.



Unfortunately there does not exist a circuit, but it is possible to connect the chains, be it with the introduction of one non-phonetic-error-proof link. These improper links are shown in the drawing by the dotted lines. The starting points 058 and 174 are obtainable by taking for the initial permutation f_0 the permutations $h_0(x+3)$ and $h_1(x+8)$ respectively. The first one happens to be the same as the one required for making an improper link. A natural choice for an infinite sequence of g's is therefore: $h_0(x+3), h_2(x+2), h_3(x+5), h_2(x+5), h_1(x+2), h_0(x+5)$, where these six permutations have to be repeated in this order. The resulting permutations f_i become: $f_0(x)=h_0(x+3)$, $f_1(x)=h_2(h_0(x+3)+2)$, $f_2(x)=h_3(h_2(h_0(x+3)+2)+5)$ and so on. In the table on the next page the various permutations are given.

x	0	1	2	3	4	5	6	7	8	9
$h_0(x+3)$	5	8	1	3	6	9	7	0	4	2
$h_2(x+2)$	9	6	4	1	8	5	7	3	0	2
$h_3(x+5)$	1	7	9	6	3	0	8	5	2	4
$h_2(x+5)$	1	8	5	7	3	0	2	9	6	4
$h_1(x+2)$	2	5	3	6	9	1	4	7	0	8
$h_0(x+5)$	1	3	6	9	7	0	4	2	5	8
$h_0(x+3)$	5	8	1	3	6	9	7	0	4	2
$h_2(x+2)$	9	6	4	1	8	5	7	3	0	2
$h_3(x+5)$	1	7	9	6	3	0	8	5	2	4
$h_2(x+5)$	1	8	5	7	3	0	2	9	6	4

x	0	1	2	3	4	5	6	7	8	9
$f_0(x)$	5	8	1	3	6	9	7	0	4	2
$f_1(x)$	5	0	6	1	7	2	3	9	8	4
$f_2(x)$	0	1	8	7	5	9	6	4	2	3
$f_3(x)$	1	8	6	9	0	4	2	3	5	7
$f_4(x)$	5	0	4	8	2	9	3	6	1	7
$f_5(x)$	0	1	7	5	6	8	9	4	3	2
$f_6(x)$	5	8	0	9	7	4	2	6	3	1
$f_7(x)$	5	0	9	2	3	8	4	7	1	6
$f_8(x)$	0	1	4	9	6	2	3	5	7	8
$f_9(x)$	1	8	3	4	2	5	7	0	9	6

This code detects 97.8% of the transpositions and the twin errors, 95.6% of the jump transpositions and the jump twin errors and about 97.9% of the phonetic errors. The latter percentage will be slightly better for short codes, since one out of the six positions will miss one of the eight possible phonetic errors.

The period of the code above is 90, since after 15 repetitions the permutations will be reproduced. This follows from the fact that the cycle representations of f_5 is (0)(1)(27469)(583) and hence the order of f_5 is 15. The same method of search might be applied to the permutations, which give rise to codes detecting only 42 of the 45 twin errors per position. This is not elaborated here, since the next chapters contain codes which are superior anyhow. It should be emphasized however that the considerations of the chapters 3,4,5 are based on the assumption that the various errors are uniformly distributed. If e.g. a certain transposition ab→ba is rare, it would be an obvious advantage to adapt a check equation in such a way that the "missed transposition" will be such a rare one. More, reliable statistics from different sources will be needed before it is justified to follow such a strategy.

Chapter 4. Codes based on the dihedral group of order 10

4.0 Some definitions

In this chapter the idea of applying the dihedral group of order 10 will be pursued.

The cyclic group of order 10 and the dihedral group are the only groups of order 10. The latter one is generated by two elements δ and ε, with the generating relations $\delta^5 = \delta^0$; $\varepsilon^2 = \varepsilon^0$; $\delta\varepsilon = \varepsilon\delta^{-1}$.

The group is denoted by D_5, since it is a transformation group of the pentagon. The δ stands for the rotations over 72 degrees, whereas the ε stands for the transformation which turns the plane of the pentagon upside down. The generating relations will be self evident in this interpretation. The elements of the group represent the symmetries of the pentagon, i.e. δ^j, with $1 \leq j \leq 4$ are the rotation symmetries and $\delta^j\varepsilon$ with $0 \leq j \leq 4$ are the reflexions with respect to the 5 axes. The elements can be coded arbitrarily with the 10 decimal digits. In this chapter is chosen for the coding: $\delta^j \rightarrow j$ and $\delta^j\varepsilon \rightarrow j+5$, for $0 \leq j \leq 4$.

The dihedral group is non-abelian, as can be seen from the third generating relation. For this reason the operation will be written as a multiplication. The multiplication is denoted by the sign "$\times$", but this sign is often omitted when the generating elements are multiplied, as in $\delta^j\varepsilon$.

In the table on the next page the result of the multiplication is given. This table, sometimes called Cayley table, can be taken as an alternative definition of the group $(D, \times)$.

Here, as in chapter 3, D stands for the set $\{1,2,3,4,5,6,7,8,9,0\}$.

Note that the digit 0 denotes the multiplicative unit of D_5.

The group D_5 has as a subgroup $(\{0,1,2,3,4\},\times)$ which is a cyclic group of the fifth order, C_5. Also $(\{0,5\},\times)$ is a cyclic subgroup (C_2).

The Cayley table of D_5.

×	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	0	6	7	8	9	5
2	2	3	4	0	1	7	8	9	5	6
3	3	4	0	1	2	8	9	5	6	7
4	4	0	1	2	3	9	5	6	7	8
5	5	9	8	7	6	0	4	3	2	1
6	6	5	9	8	7	1	0	4	3	2
7	7	6	5	9	8	2	1	0	4	3
8	8	7	6	5	9	3	2	1	0	4
9	9	8	7	6	5	4	3	2	1	0

The unit, 0, has the order 1 and the elements 1, 2, 3, 4 have the order 5, since $\delta^i = i$ for $1 \leq i \leq 4$ and $(\delta^i)^5 = \delta^{5i} = \delta^0$. The 5 elements 5, 6, 7, 8, 9 are all of the order 2 since $\delta^j\varepsilon\delta^j\varepsilon = \delta^j\delta^{-j}\varepsilon^2 = \delta^0\varepsilon^0 = \delta^0$. This was to be expected from the geometrical interpretation of the group, since reflexions are of the second order.

The automorphisms of a group leave the order of the elements invariant. Moreover an automorphism is determined as soon its effect on the generators is known. For the first generator δ, which is of the fifth order, there are 4 possible images. For the second generator ε, which is of the second order, there are 5 images feasable. The total number of automorphisms is 20, since all these combinations are admitted. The 10 inner automorphisms r_a defined by $r_a(x) = a \times x \times a^{-1}$ form a subgroup of the group of all automorphisms. The automorphism group of D_5 will in this chapter be denoted by A. A is generated by two elements ρ and σ, with the relations $\rho^5 = \rho^0$; $\sigma^4 = \sigma^0$; $\sigma\rho = \rho^2\sigma$.

The cycle representation of ρ and σ are respectively:

(0)(1)(2)(3)(4)(56789) and (0)(1243)(5)(6798).

The powers of σ form a subgroup of the order 4.

The elements of A are permutations of D and hence $A \subseteq S$. The permutations of A are listed in the table below.

0	1	2	3	4	5	6	7	8	9	
0	1	2	3	4	5	6	7	8	9	ρ^0
0	1	2	3	4	6	7	8	9	5	ρ^1
0	1	2	3	4	7	8	9	5	6	ρ^2
0	1	2	3	4	8	9	5	6	7	ρ^3
0	1	2	3	4	9	5	6	7	8	ρ^4
0	4	3	2	1	5	9	8	7	6	σ^2
0	4	3	2	1	6	5	9	8	7	$\rho\sigma^2$
0	4	3	2	1	7	6	5	9	8	$\rho^2\sigma^2$
0	4	3	2	1	8	7	6	5	9	$\rho^3\sigma^2$
0	4	3	2	1	9	8	7	6	5	$\rho^4\sigma^2$
0	2	4	1	3	5	7	9	6	8	σ
0	2	4	1	3	6	8	5	7	9	$\rho\sigma$
0	2	4	1	3	7	9	6	8	5	$\rho^2\sigma$
0	2	4	1	3	8	5	7	9	6	$\rho^3\sigma$
0	2	4	1	3	9	6	8	5	7	$\rho^4\sigma$
0	3	1	4	2	5	8	6	9	7	σ^3
0	3	1	4	2	6	9	7	5	8	$\rho\sigma^3$
0	3	1	4	2	7	5	8	6	9	$\rho^2\sigma^3$
0	3	1	4	2	8	6	9	7	5	$\rho^3\sigma^3$
0	3	1	4	2	9	7	5	8	6	$\rho^4\sigma^3$

The first 10 permutations form the subgroup of the inner automorphisms.

4.1 Formulation of the requirements

Analogous to the method of chapter 3, the codes based on D_5 are defined as the set of all code words from D^n satisfying
$c_0 \times f_1(a_1) \times f_2(a_2) \times \ldots \times f_n(a_n) = c_n$ for fixed $c_0, c_n \in D$ and $f_i \in S$.
It was shown in 3.0 that such a code detects all single errors. From this point on the parallel with chapter 3 is broken in the sense that the results are different. The treatment as a whole is analogous and in fact, it is possible to formulate some of the proofs in such a way, that they are valid for both the cyclic and the dihedral group. For didactical reasons it was thought better to make both chapters self-

contained. Especially for those readers not familiar with group theory, the third chapter would seem to be unnecessarily complicated.
The condition for the detection of the transpositions is
$c_0 \times f_1(a_1) \times \ldots \times f_i(a_i) \times f_{i+1}(a_{i+1}) \times \ldots \neq c_0 \times \ldots \times f_i(a_{i+1}) \times f_{i+1}(a_i) \times \ldots$,
for $a_i \neq a_{i+1}$.
The common factors on both sides can be cancelled by multiplication from the left or from the right by the inverse of those factors, but in the resulting inequality: $f_i(a_i) \times f_{i+1}(a_{i+1}) \neq f_i(a_{i+1}) \times f_{i+1}(a_i)$ the factors containing a_i and the ones containing a_{i+1} cannot be separated. Substitution of x for $f_i(a_i)$ and y for $f_i(a_{i+1})$ gives:
$x \times f_{i+1} f_i^{-1}(y) \neq y \times f_{i+1} f_i^{-1}(x)$ for all $x, y \in D$ with $x \neq y$.
The condition for the detection of the twin errors will become after the same reasoning: $x \times f_{i+1} f_i^{-1}(x) \neq y \times f_{i+1} f_i^{-1}(y)$ for all $x, y \in D$ with $x \neq y$. In the latter condition the x and the y are separated.
The condition for the jump transpositions becomes
$f_i(a_i) \times f_{i+1}(a_{i+1}) \times f_{i+2}(a_{i+2}) \neq f_i(a_{i+2}) \times f_{i+1}(a_{i+1}) \times f_{i+2}(a_i)$ or after the substitutions $f_i(a_i) \to x$; $f_{i+1}(a_{i+1}) \to y$; $f_i(a_{i+2}) \to z$
$x \times y \times f_{i+2} f_i^{-1}(z) \neq z \times y \times f_{i+2} f_i^{-1}(x)$ for all $x, y, z \in D$ with $x \neq z$. In the same way the condition for the detection of the jump twin errors will become: $x \times y \times f_{i+2} f_i^{-1}(x) \neq z \times y \times f_{i+2} f_i^{-1}(z)$ for all $x, y, z \in D$ with $x \neq z$.
Both conditions for the jump errors are exacting, since they have to hold for all y. It will be seen in section 4.5 that these exacting functional relations not only ask much, but also give much (see also 11).
The phonetic errors are detected if $f_i(x) \times f_{i+1}(0) \neq f_i(1) \times f_{i+1}(x)$ for all $x \neq 0,1$ with $x \in D$. Since for 1 and 0 the inequality is valid anyhow the provision $x \neq 0,1$ may be dropped.
The permutations f_i occurring in the check equation may be defined recursively by $f_{i+1} = g_i f_i$, with $f_1 \in S$.
The permutations g_i are used in the summary of the conditions.

1) Transpositions $x \times g_i(y) \neq y \times g_i(x)$ for $x, y \in D$ with $x \neq y$.
2) Twin errors $x \times g_i(x) \neq y \times g_i(y)$ for $x, y \in D$ with $x \neq y$.
3) Jump transpositions $x \times y \times g_{i+1} g_i(z) \neq z \times y \times g_{i+1} g_i(x)$ for $x, y, z \in D$ with $x \neq z$.

4) Jump twin errors $x \times y \times g_{i+1} g_i(x) \neq z \times y \times g_{i+1} g_i(z)$ for $x, y, z \in D$ with $x \neq z$.

5) Phonetic errors $x \times f_{i+1}(0) \neq f_i(1) \times g_i(x)$ for $x \in D$.

4.2 The analysis of the conditions.

The fifth and the first condition are no longer contradictory as they were in the cyclic case.

The proof of the impossibility of the first four conditions is not applicable, since D_5 is not abelian. In fact, and this is the advantage of the dihedral group, there do exist permutations which satisfy the first condition (see 4.4). The twin error detection requires that $x \times g(x)$ is a permutation for some permutation g. Although the variables are separated in the twin error condition, the non-existence proof of 3.2 is not applicable, since $\prod_{x \in D} x = \prod_{x \in D} g(x)$ does not always hold in D_5. Unfortunately this does not imply that there exist permutations satisfying the requirement.

Theorem 4.2.0. There does not exist a permutation f, such that $x \times f(x)$ is also a permutation.

Proof: Let the digits 0, 1, 2, 3, 4 be called low and the remaining digits high, lo and hi for short. Suppose that f(x) is k times low for low x. Thus k times: f(lo) = lo and hence 5-k times: f(lo) = hi and f(hi) = lo, and thus k times f(hi) = hi. The low digits form a subgroup of D_5 with a factor group of order 2, which means that lo×lo = hi×hi = lo and lo×hi = hi×lo = hi. From this it follows that $x \times f(x)$ is 2k times lo and 10-2k times hi. If $x \times f(x)$ were a permutation then 5 low and 5 high digits had to occur, but 2k = 5 is not true. Hence $x \times f(x)$ can at best be nearly a permutation. The following example shows that this is indeed possible.

x	0	1	2	3	4	5	6	7	8	9
f(x)	0	2	4	6	7	5	9	8	3	1
x×f(x)	0	3	1	9	6	0	2	4	5	8

This particular f does not satisfy the first condition, since $0\times f(5) = 5\times f(0) = 5$. This is not accidental, since it can be shown that no permutation f exists, such that $x\times f(x)$ is nearly a permutation and such that it also satisfies the condition that $x\times f(y) \neq y\times f(x)$ for all $x \neq y$. The proof which is very cumbersome, distinguishing several cases, is left out. The fact will be established by the computer search anyhow, which in itself is also a proof, distinguishing all cases.
It is also possible to give an upper bound for the detection rate of the jump errors. There are 450 combinations for $x,y,z \in D$ with $x < z$. Let $g_{i+1}g_i$ be denoted by h for short. The conditions then become $x\times y\times h(z) \neq z\times y\times h(x)$ and $x\times y\times h(x) \neq z\times y\times h(z)$. As in the proof of theorem 4.2.0, the pairs $(x,h(x))$ can be put into four classes denoted. by A_{ij}, with $i,j \in \{0,1\}$. The i and the j can be taken as the exponent of ε of x and h(x) respectively. So if $x = \delta^k\varepsilon^i$ and $h(x) = \delta^l\varepsilon^j$ then $(x,h(x)) \in A_{ij}$. Obviously $|A_{00}| + |A_{01}| + |A_{10}| + |A_{11}| = 10$ and $|A_{00}| + |A_{01}| = |A_{10}| + |A_{11}| = |A_{00}| + |A_{10}| = |A_{01}| + |A_{11}| = 5$.
Two different classes A_{ij} and A_{kl} are called complementary if $i+j = k+l \pmod 2$. Three cases are considered separately.
i) $(x,h(x))$ and $(z,h(z))$ belong to different non-complementary classes. Both inequalities are then trivially fulfilled, since the number of ε's is different on both sides of the sign, no matter what value y has.
ii) $(x,h(x))$ and $(z,h(z))$ belong to the same class, say A_{ij}. Suppose that $x = \delta^a\varepsilon^i$; $z = \delta^c\varepsilon^i$; $y = \delta^b\varepsilon^k$; $h(x) = \delta^{a'}\varepsilon^j$; $h(z) = \delta^{c'}\varepsilon^j$ express the representations of the various elements as products of the generators of D_5. The conditions become after substitution:
$\delta^a\varepsilon^i\delta^b\varepsilon^k\delta^{c'}\varepsilon^j \neq \delta^c\varepsilon^i\delta^b\varepsilon^k\delta^{a'}\varepsilon^j$ and $\delta^a\varepsilon^i\delta^b\varepsilon^k\delta^{a'}\varepsilon^j \neq \delta^c\varepsilon^i\delta^b\varepsilon^k\delta^{c'}\varepsilon^j$ which can be converted, using the generating relations of D_5, into

$$\delta^{a+(-1)^i b+(-1)^{i+k}c'}\varepsilon^{i+j+k} \neq \delta^{c+(-1)^i b+(-1)^{i+k}a'}\varepsilon^{i+j+k} \text{ and}$$

$$\delta^{a+(-1)^i b+(-1)^{i+k}a'}\varepsilon^{i+j+k} \neq \delta^{c+(-1)^i b+(-1)^{i+k}c'}\varepsilon^{i+j+k}.$$

After cancelling ε^{i+j+k} it follows that:

$$a+(-1)^i b+(-1)^{i+k}c' \neq c+(-1)^i b+(-1)^{i+k}a' \pmod 5 \text{ and}$$

$$a+(-1)^i b+(-1)^{i+k}a' \neq c+(-1)^i b+(-1)^{i+k}c' \pmod 5.$$

These inequalities are independent of b and they can be reduced to: $a-c \neq (-1)^{i+k}(a'-c')$ and $a-c \neq (-1)^{i+k}(c'-a')$. For 5 out of the 10 values for y it holds that $(-1)^{i+k} = 1$, and for the other 5 values $(-1)^{i+k} = -1$. The 2 conditions are split up into $a-c \neq a'-c' \pmod 5$ and $a-c = -(a'-c') \pmod 5$ for the first condition and $a-c = c'-a' \pmod 5$ and $a-c = -(c'-a') \pmod 5$ for the second one. The 2 pairs of inequalities are apparently equivalent. From $x \neq z$ it follows in this case that $a \neq c \pmod 5$ and hence $a-c = a'-c' \pmod 5$ and $a-c = -(a'-c') \pmod 5$ cannot both hold at the same time. Thus if one of the two holds, then the original inequalities are both valid for 5 of the 10 values for y and otherwise for all 10 values.

iii) $(x,h(x))$ and $(z,h(z))$ belong to complementary classes, say A_{ij} and $A_{1-i,1-j}$ respectively. Since either i or 1-i is equal to 0 and since the conditions are symmetric with respect to x and z, it may be assumed that $i = 0$ holds. Suppose that $x = \delta^{a}$; $y = \delta^{b}\varepsilon^{k}$; $z = \delta^{c}\varepsilon$; $h(x) = \delta^{a'}\varepsilon^{j}$; $h(z) = \delta^{c'}\varepsilon^{1-j}$ holds. Substitution in the conditions gives $\delta^{a}\delta^{b}\varepsilon^{k}\delta^{c'}\varepsilon^{1-j} \neq \delta^{c}\varepsilon\delta^{b}\varepsilon^{k}\delta^{a'}\varepsilon^{j}$ and $\delta^{a}\delta^{b}\varepsilon^{k}\delta^{a'}\varepsilon^{j} \neq \delta^{c}\varepsilon\delta^{b}\varepsilon^{k}\delta^{c'}\varepsilon^{1-j}$ which becomes after setting the generating relation at work

$$\delta^{a+b+(-1)^{k}c'}\varepsilon^{1-j+k} \neq \delta^{c-b+(-1)^{k+1}a'}\varepsilon^{1+k+j} \text{ and}$$

$$\delta^{a+b+(-1)^{k}a'}\varepsilon^{j+k} \neq \delta^{c-b+(-1)^{k+1}c'}\varepsilon^{1+k+1-j}.$$

Since the exponents of ε is the same modulo 2 on both sides of both inequalities it is necessary that the exponents of δ are different modulo 5. That is $a+b+(-1)^{k}c' \neq c-b-(-1)^{k}a' \pmod 5$ and $a+b+(-1)^{k}a' \neq c-b-(-1)^{k}c' \pmod 5$. After sorting the terms both equations can be put in the form $2b \neq c-a-(-1)^{k}(c'+a') \pmod 5$. Hence, for each of the two values for k, there is just one of the 5 values for b such that the inequality is false.

The following table gives a survey of the number of undetected jump errors.

	A_{00}	A_{01}	A_{10}	A_{11}
A_{00}	0 or 5	0	0	2
A_{01}	0	0 or 5	2	0
A_{10}	0	2	0 or 5	0
A_{11}	2	0	0	0 or 5

The best permutations would be those which score always a 0 in the main diagonal. The number of triplets x,y,z, with $z > x$, which fall in the complementary classes is $d^2+(5-d)^2$, with $d = |A_{00}|$. Hence out of the 450 cases $2(d^2+(5-d)^2)$ are bound to remain undetected. The function $d^2+(5-d)^2$ is at least 13 so that at most 424 of the 450 possible jump errors are detected. This would be a detection rate of 94.2%. It will be seen in 4.5 that there exist permutations which achieve this reault.

As to the phonetic errors it is sufficient to remark that the detecting condition does not contradict the one for the transpositions, so that a 100% detection seems feasible. Section 4.6 is devoted to the construction of codes which reach that score.

4.3 Detection rate preserving transformations

Let U be the set of all permutations f, satisfying $x \times f(y) \neq y \times f(x)$ for all $x,y \in D$ with $x \neq y$ and let V be the subset of U consisting of the permutations f, such that $x \times f(x) = y \times f(y)$ is valid for 2 or less (unordered) pairs x,y with $x \neq y$.

As in the cyclic case there are several transformations (of S) which leave U and V invariant, but the situation is more complicated, since D_5 is non-abelian and since the concept of duplicators cannot be used. From the fact that D_5 is a group it follows that multiplication from the right (or the left) by a fixed element a, permutes the elements of D. Let the resulting permutations be denoted by r_a and l_a respectively. Hence $r_a(x) = x \times a$ and $l_a(x) = a \times x$. The permutations l_a with $a \in D$ form a subgroup of the symmetric group S, called the left regular representation of D_5. This subgroup is isomorphic with D_5, since from

$l_a l_b(x) = l_a(b\times x) = a\times(b\times x) = (a\times b)\times x = l_{a\times b}(x)$ it follows that $l_a l_b = l_{a\times b}$ and since $l_a \neq l_b$ for $a \neq b$. The 10 different permutations r_a, with $a \in D$, form also a subgroup of S, called the right regular representation of D_5. This group is also isomorphic with D_5, since from $r_a r_b(x) = r_a(x\times b) = x\times b\times a = r_{b\times a}(x)$, it follows that $r_a r_b = r_{b\times a}$, so that the order of multiplication is reversed. The transformations $R_a: f \to r_a f$ leave U and V invariant, since from $x\times f(y) = z\times f(u)$ it follows that $x\times f(y)\times a = x\times r_a f(y) = z\times r_a f(u) = z\times f(u)\times a$. Hence if $f \in U,V$ then also $r_a f \in U,V$. The induced equivalence classes all have 10 different elements and in each class a representative, which has 0 as a fixed point, can be selected. The permutation $r_{f(0)^{-1}} f$ is equivalent with f and has 0 as a fixed point. Let U_0 and V_0 be defined by $U_0 = \{f \mid f(0) = 0,\ f \in U\}$ and $V_0 = \{f \mid f(0) = 0,\ f \in V\}$. Evidently $|U_0| = |U|/10$ and $|V_0| = |V|/10$ hold. The search for permutations satisfying the conditions for the detection of the transpositions and the twin errors may be limited by setting $f(0) = 0$.

The transformation $f \to l_a f$ does not leave U invariant, as the following counterexample shows. The permutation f given by $f = \binom{0123456789}{0432178956}$ belongs to U, as will be seen later on. $l_5 f$ however does not belong to U since $5\times l_5 f(8) = 5\times 5\times f(8) = 5$ and $8\times l_5 f(5) = 8\times 5\times f(5) = 3\times 7 = 5$.

The transformation $L_a: f \to f l_a$ leaves both U and V invariant. This follows at once by substituting $a\times x$; $a\times y$ etc. in $x\times f(y) \neq z\times f(u)$ which gives $a\times(x\times f l_a(y)) \neq a\times(z\times f l_a(u))$. Since $f l_a(0) = f(a)$ it is clear that U_0 is not invariant for all L_a.

The transformations R_b and L_a are permutable since $R_b L_a(f) = R_b(f l_a) = r_b f l_a = L_a(r_b f) = L_a R_b(f)$ holds.

It is possible to construct a transformation T_a such that $T_a(U_0) = U_0$. Define T_a by $T_a(f) = L_a R_{f(a)^{-1}}(f)$ and let $T_a(f) = g$, then $g(0) = f(a\times 0)\times f(a)^{-1} = 0$. Moreover if $T_b g = h$ then $g(x) = f(a\times x)\times f(a)^{-1}$ and $h(x) = g(b\times x)\times g(b)^{-1} = f(a\times b\times x)\times f(a)^{-1}\times(f(a\times b)\times f(a)^{-1})^{-1} = f(a\times b\times x)\times f(a\times b)^{-1}$ and hence $h = T_{a\times b}$. Thus $T_b T_a = T_{a\times b}$ is valid. Furthermore $T_0(f) = f$, since $f(x) = f(0\times x)\times f(0)^{-1} = f(x)$ provided that $f(0) = 0$ holds. The transformations T_a, with $a \in D$, working on U_0,

form therefore a group. The equivalence classes induced do however not always contain 10 permutations, as the following example will show. Define f by $f(\delta^k) = \delta^{-k}$ and $f(\delta^j\varepsilon) = \delta^{j+d}\varepsilon$, with $d \neq 0$, then $f(x) = f(5\times x)\times f(5)^{-1}$ for $x=\delta^k$; $f(\varepsilon\times\delta^k)\times f(\varepsilon)^{-1} = f(\delta^{-k}\varepsilon)\times(\delta^d\varepsilon)^{-1} =$ $= \delta^{-k}\delta^d\varepsilon(\delta^d\varepsilon)^{-1} = \delta^{-k} = f(\delta^k)$ and for $x=\delta^j\varepsilon$:$(\varepsilon\times\delta^j\varepsilon)\times f(\varepsilon)^{-1} =$ $= f(\delta^{-j})\times\delta^d\varepsilon = \delta^{j+d}\varepsilon = f(\delta^j\varepsilon)$.

The relation $x\times f(y) \neq y\times f(x)$ is proved by treating the three cases x and y both low or both high and x and y in different classes, separately.

i) $\delta^i\times f(\delta^j) = \delta^{i-j}$ whereas $\delta^j\times f(\delta^i) = \delta^{j-i}$, but $i-j \neq 0$ since $x \neq y$.

ii) $\delta^i\varepsilon\times f(\delta^j\varepsilon) = \delta^i\varepsilon\delta^{j+d}\varepsilon = \delta^{i-j-d}$, but $\delta^j\varepsilon\times f(\delta^i\varepsilon) = \delta^j\varepsilon\delta^{i+d}\varepsilon =$ $= \delta^{j-i-d}$, and again $i-j \neq 0$.

iii) $\delta^i\times f(\delta^j\varepsilon) = \delta^i\delta^{j+d}\varepsilon = \delta^{i+j+d}\varepsilon$ and $\delta^j\varepsilon\times f(\delta^i) = \delta^j\varepsilon\delta^{-i} = \delta^{i+j}\varepsilon$.

Thus $f \in U$ holds, but the detection of the twin errors is bad, since $\delta^i\times f(\delta^i) = 0$ and $\delta^j\varepsilon\times f(\delta^j\varepsilon) = \delta^{-d}$. Hence 20 (i.e. $2\binom{5}{2}$) of the 45 possible twin errors escape detection. This code will be met again in the next chapter. Clearly $f \notin V_0$ holds and it will be seen in 4.4 that in V_0 the equivalence classes do contain 10 elements each. Meanwhile it is not clear how this transformation can be used in the search for U_0. The third transformation group is of a different nature. It is a subgroup of the automorphism group of S, consisting of the inner automorphisms derived from the elements of A, which is, as automorphism group of D_5, a subgroup of S. The fact, which will be proved below, that the transforms by elements of A, leave U and V invariant means that A is contained in the normalizer of U and V. The transformations are denoted by F_s, with $s \in A$, and defined by $F_s(f) = sfs^{-1}$ for $f \in S$. The transformations form a group isomorphic with A, since $F_sF_t(f) = F_s(tft^{-1}) = stft^{-1}s^{-1} = (st)f(st)^{-1} = F_{st}(f)$ and since $F_s \neq F_t$ for $s \neq t$.

From $x\times f(y) \neq y\times f(x)$ it follows that $s(x\times f(y)) \neq s(y\times f(x))$ and as s is an automorphism, the inequality becomes after substituting $s^{-1}x'$ for x and $s^{-1}y'$ for y: $x'\times sf(s^{-1}y') \neq y'\times sf(s^{-1}x')$. Hence $F_s(f) \in U$ if $f \in U$. The same kind of reasoning proves that $F_s(f) \in V$ if $f \in V$.

Since the automorphisms leave 0 fixed, it follows that the transformations F_s leave U_0 and V_0 invariant as well.
The induced equivalence classes do not necessarily have 20 elements each. It can happen that $sfs^{-1} = f$, or what is the same, that $sf = fs$. All the automorphisms except those belonging to the subgroup C_5, have just one fixed point different from 0, say $s(a) = a$. Therefore, if $sf = fs$ then $sf(a) = fs(a) = f(a)$ and hence $f(a) = a$. But then $a \times f(0) = a = 0 \times f(a)$ holds, and thus $f \notin U_0$. The permutations permutable with the subgroup C_5 are of the form $\rho^i f'$, where f' is a permutation which leaves each one of the high digits fixed, as can be readily verified (see 8). In fact the example given above is of this very form. These permutations give a poor detection of the twin errors.
The latter property follows from the proof of theorem 4.2.0, since $\rho^i f'(lo) = lo$ and $\rho^i f'(hi) = hi$, so that $x \times f(x) = lo$ for all x. The detection rate is therefore at most 40/45 and $\rho^i f'$ does not belong to V.
For the search for U_0 only a subgroup of A is useful, namely C_4 consisting of the automorphisms σ^j, with $0 \leq j \leq 3$. With this group a factor 4 can be gained. Let U_{0i} and V_{0i} be defined by $U_{0i} = \{f \mid f \in U_0,\ f(5) = i\}$ and $V_{0i} = \{f \mid f \in V_0,\ f(5) = i\}$. Now the search may be limited to say the classes U_{03} and U_{08} for, if $f(5) \in \{1,2,3,4\}$ then j can be chosen such that $\sigma^j f \sigma^{-j}(5) = 3$ and if $f(5) \in \{6,7,8,9\}$ then for some j: $\sigma^j f \sigma^{-j}(5) = 8$. Note that $f(5) \neq 5$ if $f \in U$.

4.4 The search program

The same program as in the preceding chapter can be used, except for a few changes. First of all the group operation has to be adapted to the dihedral group. Secondly the test has to be changed. The program is used twice i.e. once for U_{03}, in which $f(5) = 3$ and the other time for U_{08} with $f(5) = 8$. The result of the 90 seconds search is that $|U_{03}| = 404$ and $|U_{08}| = 447$, so that $|U| = 34040$. There are no permutations in U for which $x \times f(x)$ is nearly a permutation. Furthermore $|V_{03}| = 72$ and $|V_{08}| = 78$ which makes $|V| = 6000$. It turns out that the transformations T_a divide V into 600 classes and that the transformations F_s with $s \in A$ and R_a with $a \in D$ give a further subdivision of these

classes. In V three permutations h_1, h_2 and h_3 can be chosen such that each $f \in V$ can be written as $L_a R_b F_s h_i$ with $a,b \in D$; $s \in A$ and $i \in \{1,2,3\}$. The three permutations h_i are given in the table below.

x	0	1	2	3	4	5	6	7	8	9
$h_1(x)$	0	6	4	2	7	8	1	3	9	5
$h_2(x)$	0	5	8	2	6	3	7	9	4	1
$h_3(x)$	0	7	9	6	1	8	4	2	3	5

The importance of this canonical representation will become clear in the next sections.

4.5 The detection of the jump errors

The requirement for the optimal detection of the jump transpositions and the jump twin errors is the same, as was shown in section 2 of this chapter. Let g and g' be two permutations of V, not necessarily different. It will be said that g matches g' if $x \times y \times gg'(x) \neq z \times y \times gg'(z)$ holds for 424 out of the 450 triplets x,y,z with $x,y,z \in D$ and $z > x$. For a "good" code a chain of matching g_i's is needed in order to construct the f_i's recursively with $f_{i+1} = g_i f_i$ and $f_1 \in S$. Just as in chapter 3 it is advisable to make a catalogue of the matching pairs as a preparation for the construction of the chains.
In view of the large number of possible pairs (36000000), other reason could be mentioned as well, it is worthwile to exploit the equivalence transformations of section 4.3. As a matter of fact it will turn out that a factor 20000 can be gained on the number of tests to be performed. To avoid unnecessarily complicated formulae the proof will be given in several steps.
The transformations L_a, R_b and F_s with $a,b \in D$ and $s \in A$ satisfy the following relations: 0) $L_a L_b = L_{b \times a}$; 1) $R_a R_b = R_{b \times a}$; 2) $F_s F_t = F_{st}$; 3) $L_a R_b = R_b L_a$; 4) $F_s R_b = R_{s(b)} F_s$; 5) $F_s L_a = L_{s(a)} F_s$.
The first four relations have been proved in 4.3. The remaining two follow directly from the definitions, for $F_s R_b(f) = s(r_b f)s^{-1}$ and for each $x \in D$, $sr_b fs^{-1}(x) = s(fs^{-1}(x) \times b) = sfs^{-1}(x) \times s(b)$ and hence

4) is proved. Also $F_sL_a(f) = s(fl_a)s^{-1}$ and for each $x \in D$ $sfl_as^{-1}(x) =$ $= sf(a\times s^{-1}(x)) = sfs^{-1}(s(a)\times x)$ and relation 5 follows at once.

Denote the number of solutions of $x\times y\times gg'(x) = z\times y\times gg'(z)$ with $z > x$ and $x,y,z \in D$, by $N(g,g')$. The next step is to prove the relations:

6) $N(g,g') = N(R_bg,g')$; 7) $N(g,g') = N(g,L_ag')$; 8) $N(g,F_sg') =$ $= N(F_{s^{-1}}g,g')$; 9) $N(L_ag,g') = N(F_{r^{-1}}g,R_ag')$ where r is the inner automorphism defined by $r(x) = a\times x\times a^{-1}$.

Consider the equation $x\times y\times gg'(x) = z\times y\times gg'(z)$ and multiply both sides from the right by b to prove the relation 6. Substitution of $a\times x'$ for x and $a\times z'$ for z followed by multiplication from the left of both sides by a^{-1} gives relation 7. Now consider $x\times y\times gsg's^{-1}(x) = z\times y\times gsg's^{-1}(z)$ and apply the automorphism s^{-1} on both sides and substitute $s(y')$ for y, $s(x')$ for x and $s(z')$ for z and relation 8 results. The awkward factor y in the middle proved to be helpful in this situation, since y may be replaced by $s(y)$. Finally the equation $x\times y\times g(a\times g'(x)) = z\times y\times g(a\times g'(z))$ can be altered into $x\times y\times a\times a^{-1}\times g(a\times(g'(x)\times a)\times a^{-1})\times a =$ $= z\times y\times a\times a^{-1}\times g(a\times(g'(z)\times a)\times a^{-1})\times a$ or $x\times y\times a\times r^{-1}gr(g'(x)\times a) =$ $= z\times y\times a\times r^{-1}gr(g'(z)\times a)$ which gives relation 9 after substituting $y'\times a^{-1}$ for the helpful y. By means of the relations above it is easy to prove that $N(L_aR_bF_sf_i,L_cR_dF_tf_j) = N(F_{t^{-1}r^{-1}s}f_i,R_{t^{-1}(d\times a)}f_j)$.

The L_c and the R_b can be removed by 7, 3 and 6, giving $N(L_aR_bF_sf_i,L_cR_dF_tf_j) = N(L_aF_sf_i,R_dF_tf_j)$. Application of 9 gives $N(L_aF_sf_i,R_dF_tf_j) = N(F_{r^{-1}}F_sf_i,R_aR_dF_tf_j)$. The F_t can be moved over R by 1 and 4 and after that, 8 gives $N(F_{r^{-1}}F_sf_i,R_aR_dF_tf_j) =$ $= N(F_{t^{-1}}F_{r^{-1}}F_sf_i,R_{t^{-1}(d\times a)}f_j)$. Finally application of 2 gives the desired equality.

Consequently it is only necessary to test the 1800 pairs F_sh_i, R_ah_j with $s \in A$, $a \in D$ and $i,j \in \{1,2,3\}$.

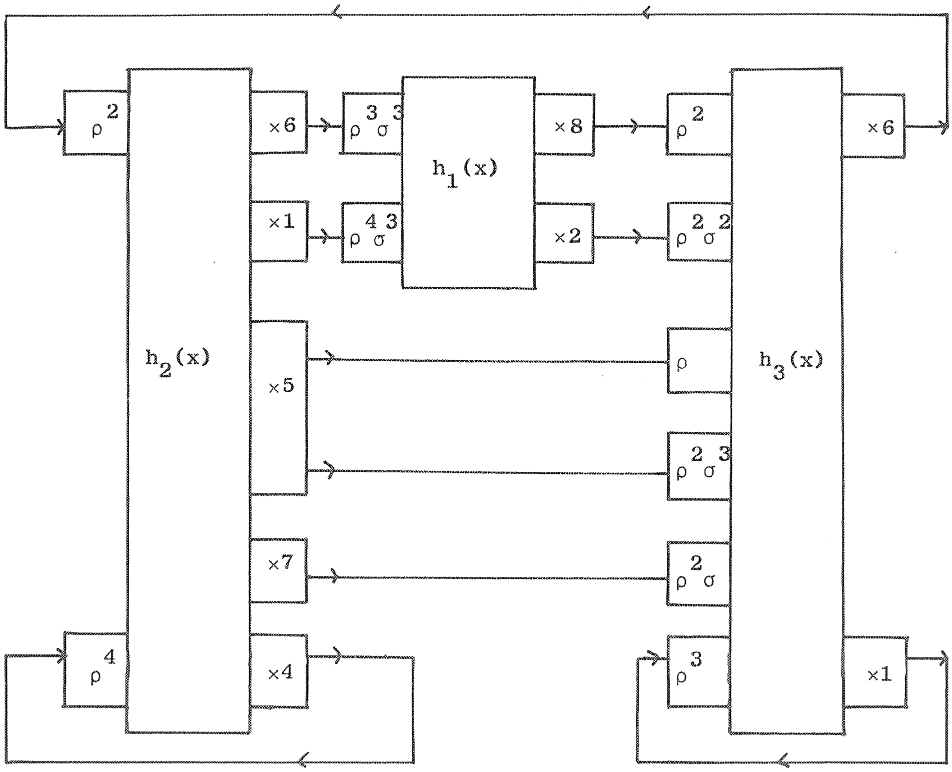
If one matching pair has been found for certain s', a', i, j then all 20000 pairs $L_aR_bF_sh_i$, $L_cR_dF_th_j$ with $t^{-1}r^{-1}s = s'$, $t^{-1}(d\times a) = a'$, with $r(x) = a\times x\times a^{-1}$, for $x \in D$, will match. The result of the test program is that 10 pairs of the form $F_{s'}h_i$, $R_{a'}h_j$ match. Let the pairs be

denoted by the quartet (s',i,j,a). The 10 pairs are $(\rho^2,3,1,8)$; $(\rho^2\sigma^2,3,1,2)$; $(\rho^4,2,2,4)$; $(\rho,3,2,5)$; $(\rho^2\sigma,3,2,7)$; $(\rho^2\sigma^3,3,2,5)$; $(\rho^3\sigma^3,1,2,6)$; $(\rho^4\sigma^3,1,2,1)$; $(\rho^3,3,3,1)$; $(\rho^2,2,3,6)$.
The diagram on the next page pictures the matching relations. It is now a simple matter to construct codes which give an optimal detection for the transpositions (100%), the twin errors (95.5%), the jump transpositions (94.2%) and the jump twin errors (94.2%). A chain of permutations g_i can be made by following the arrows in the diagram and performing the operations as indicated along the lines.
Let $L_aR_bF_sh_i$ match $L_cR_dF_th_j$ according to the quartet (s',i,j,a'), then $d\times a = t(a')$ and $s = rts'$, with $r(x) = a\times x\times a^{-1}$. By selecting d so that $d = t(a')$, it follows that $a = 0$ and $r = \rho^0$ and hence $s = ts'$.
With the following scheme a chain can be forged easily:

k	i	j	quartet	t	t(a') = d	ts' = s	g_k
1	2	3	$(\rho,3,2,5)$	ρ^0	$\rho^0(5) = 5$	$\rho^0\rho = \rho$	R_5h_2
2	3	3	$(\rho^3,3,3,1)$	ρ	$\rho(1) = 1$	$\rho\rho^3 = \rho^4$	$R_1F_\rho h_3$
3	3	2	$(\rho^2,2,3,6)$	ρ^4	$\rho^4(6) = 5$	$\rho^2\rho^4 = \rho$	$R_5F_{\rho^4}h_3$
4	2	1	$(\rho^3\sigma^3,1,2,6)$	ρ	$\rho(6) = 7$	$\rho\rho^3\sigma^3 = \rho^4\sigma^3$	$R_7F_\rho h_2$

etc.

Of special interest are the loops by h_2 and h_3, since they offer the possibility to form progressive codes which have all g_i's equal. So can $F_sL_aR_bh_2$ be self-matching if $s^{-1}r'^{-1}s = \rho^4$ and $b\times a = 4$, where r' is defined by $r'(x) = s(a)\times x\times s(a^{-1})$. Hence $s^{-1}r'^{-1}s(x) =$
$= s^{-1}(s(a^{-1})\times s(x)\times s(a) = a^{-1}\times x\times a$ and it follows that $a = 3$ and $b = 1$, so that $F_sL_3R_1h_2$ is self matching for all s. Now $L_3R_1h_2 =$
$= L_3R_1\binom{0123456789}{0582637941} = L_3\binom{0123456789}{1973546802} = \binom{0123456789}{3519702468} = (03986215)(47)$.
The loop by h_3 gives rise to the self matching permutations $F_tL_1h_3$, where $L_1h_3 = \binom{0123456789}{7961042358} = (07319854)(26)$. The resulting progressive codes have all a period 8. None of these codes is phonetic error-proof, but by choosing $s = \rho^2$ the permutation (01589427)(36) is obtained which provides a code with a detection rate of 95,3% for the phonetic errors.



4.6 The detection of the phonetic errors

The purpose of this section is to construct chains of matching permutations, in the sense of section 4.5, with the property that the phonetic errors are detected too. The permutations f_i occurring in the check equation $\Pi\ f_i(a_i) = c_n$, which defines the code, are given recursively by $f_{k+1} = g_k f_k$ with f_1 arbitrarily chosen in S. The permutation g_{k+1} has to match g_k for $k > 1$, but g_1 can be taken arbitrarily in U. The detecting condition $f_k(x)\times f_{k+1}(0) \neq f_k(1)\times f_{k+1}(x)$ may be written as $x\times g_k f_k(0) \neq f_k(1)\times g_k(x)$ for $x \in D$.
By putting $p = f_k(1)$ and $q = f_k(0)$ the condition becomes $x\times g_k(q) \neq$ $\neq p\times g_k(x)$ for $x \in D$. The set of pairs p,q such that the above condition is fulfilled, will be called the initial set of g_k. The new values for p and q which are offered to g_{k+1}, are $g_k(p)$ and $g_k(q)$ (or $f_{k+1}(p)$,

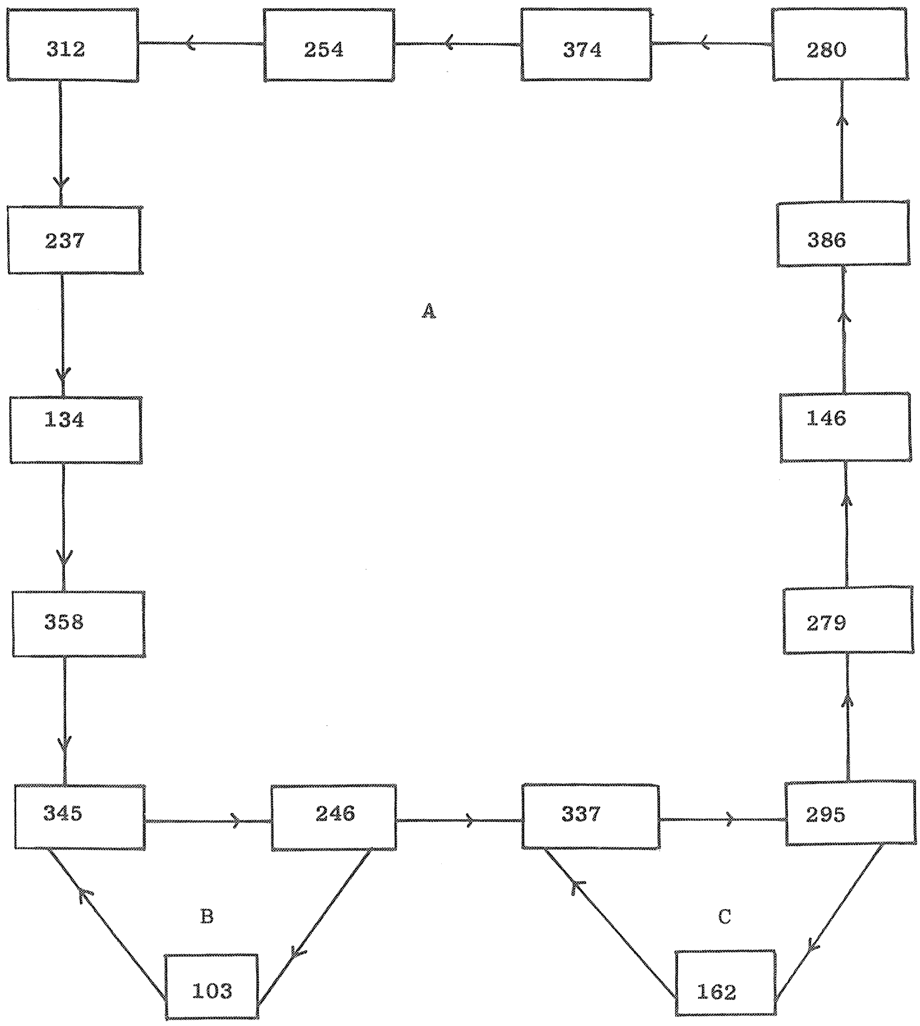
$f_{k+1}(q)$). These pairs are said to form the terminal set of g_k. The initial sets of h_1, h_2 and h_3 as defined at the end of section 4.4, can easily be found by checking the condition. Let the initial set of h_i be denoted by X_i, then the sets X_i turn out to be: $X_1 = \{01,04$ $06,07,12,13,14,19,21,27,28,38,39,43,45,51,56,63,67,68,72,73,75,78,80,87,90,91,92,94\}$; $X_2 = \{01,03,09,17,18,20,25,28,32,38,43,46,49,56,57,59,62,65,67,71,72,78,84,86,87,89,90,91,92,93,94\}$ and $X_3 = \{02,04,08,10,16,17,19,21,23,24,28,35,37,42,47,51,53,58,62,64,69,70,72,75,79,81,82,91,93,95,96\}$. Thus $|X_1| = 30$, $|X_2| = |X_3| = 31$ holds. If $g_k = L_c R_d F_t h_j$, then the initial set of g_k can be derived from the initial set of h_j. The checking condition for g_k is $x \times th_j t^{-1}(c\times q')\times d \neq$ $\neq p'\times th_j t^{-1}(c\times x)\times d$ which is equivalent with $t^{-1}(c\times x)\times t^{-1} th_j t^{-1}(c\times q') \neq$ $\neq t^{-1}(c\times p')\times t^{-1} th_j t^{-1}(c\times x)$, which after substitution of x' for $t^{-1}(c\times x)$ becomes: $x'\times h_j(t^{-1}(c\times q')) \neq t^{-1}(c\times p')\times h_j(x')$. Thus the pair $t^{-1}(c\times p')$, $t^{-1}(c\times q')$ has to belong to X_j and hence for some pair p,q from X_j, $p = t^{-1}(c\times p')$ and $q = t^{-1}(c\times q')$ has to hold. The latter equations are equivalent with $p' = c^{-1}\times t(p)$ and $q' = c^{-1}\times t(q)$ and the initial set of g_k is $c^{-1}\times t(X_j)$. The terminal set of g_k is formed by the pairs $th_j t^{-1}(c\times p')\times d$, $th_j t^{-1}(c\times q')\times d$, or $th_j(p)\times d$, $th_j(q)\times d$. The terminal set of g_k is therefore $th_j(X_j)\times d$.

Now if $F_u h_i$ matches $R_e h_j$ then g_{k+1} matches g_k if $g_{k+1} = L_a R_b F_s h_i$ with $t^{-1}(d\times a) = e$ and $t^{-1}r^{-1}s = u$, where r is again the inner automorphism defined by $r(x) = a\times x\times a^{-1}$. The phonetic errors are detected by this new link if($th_j(p)\times d$, $th_j(q)\times d)\in a^{-1}\times s(X_i)$ as is derived above. This relation implies that for certain $(p'',q'')\in X_i$ it has to hold that $th_j(p)\times d = a^{-1}\times s(p'')$ and $th_j(q)\times d = a^{-1}\times s(q'')$ are true. Substitution of rtu for s gives $th_j(p)\times d = a^{-1}\times(a\times tu(p'')\times a^{-1})$ and the similar equation for q. Substitution of e for $t^{-1}(d\times a)$ gives, since $d = t(e)\times a^{-1}$; $h_j(p) = u(p'')\times e^{-1}$ and similarly for q the equation $h_j(q) = u(q'')\times e^{-1}$. These conditions are only dependent on u and e, which depend only on the linking mode which is employed, to get from j to i, (see diagram of section 4.5). The conditions may be viewed as a directed graph K. The points of K are the triplets $(i,h_i(p),h_i(q))$, with i = 1, 2, 3 and $(p,q)\in X_i$. Now $(j,u(p)\times e^{-1},u(q)\times e^{-1})$ is connected

with $(i,h_i(p),h_i(q))$ if $F_u h_i$ matches $R_e h_j$ and if $(p,q) \in X_i$. The 10 matching modes thus give rise to 308 directed edges. A phonetic error-proof code has to be based on a chain of permutations g_k, which has to correspond with a path in the graph K. The construction is therefore brought back to the problem of finding paths , preferable circuits, in the directed graph K. The circuits are interesting since they give infinite, though periodic, codes. The circuits, if they exist, can be found by the method described in 3.5. The twigs of K can be pruned off and after 10 prunings, as it turns out, a twig-free directed graph with 35 edges is left over.

This proves that a circuit has to be present. Among these 35 edges there may be edges which have an initial vertex which is not the terminal vertex of any other edge. Edges like this may be called roots of the directed graph. The roots can be removed by the same procedure, be it that the direction of the edges has to be reversed. After cutting off all the roots, there remains a graph with 18 edges and 16 vertices. A picture is given on page 99. The graph has a rich structure, since there are three basic circuits, A, B and C, such that tours can be organized, such that A, B and C are visited in an arbitrary order, with any multiplicity. The tours are thus in a one to one correspondence with the free semi group generated by A, B and C. The circuits B and C have each three edges, whereas A has 14 edges. Most attractive, because of their simplicity, are the codes based on one of these smaller circuits only. As an example one of these codes will be constructed. It is made of the matching pairs $(F_{\rho^2} h_2, R_6 h_3)$; $(F_{\rho^2\sigma^2} h_3, R_2 h_1)$; $(F_{\rho^3\sigma^3} h_1, R_6 h_2)$. The vertices of C are (2,9,5); (1,6,2); (3,3,7). The pairs of the initial sets of h_1, h_2 and h_3 which correspond with the vertices are: (1,3); (7,1); (8,1).

Let these pairs be denoted by (p_i,q_i) and let the matching pairs be denoted by $(F_{s_1} h_2, R_6 h_3)$; $(F_{s_2} h_3, R_2 h_1)$; $(F_{s_3} h_1, R_6 h_2)$. The construction of the g-chain may begin by taking an arbitrary permutation from U as g_1. If this permutation in the canonical representation is


312
254
374
280
237
386
A
134
146
358
279
345
246
337
295
B
C
103
162

$L_cR_dF_th_j$, then the permutation f_1 has to be chosen so that $f_1(1) = c^{-1}\times t(p_j)$ and $f_1(0) = c^{-1}\times t(q_j)$, the other values of f may be chosen arbitrarily. For the matching pairs it holds that $N(F_{s_1}h_2,R_6h_3) = 26$ and by the rules 6 and 4 of section 4.5, it follows that $N(F_{s_1}h_2,R_6h_3) = N(R_{s_1}(6)F_{s_1}h_2,R_6h_3) = N(F_{s_1}R_6h_2,R_6h_3) = 26$, in an analogous way the other matching relations may be transformed. Putting $R_2h_1 = h_1'$; $R_6h_2 = h_2'$ and $R_6h_3 = h_3'$ it follows that $26 = N(F_{s_1}h_2',h_3') = N(F_{s_2}h_3',h_1') = N(F_{s_3}h_1',h_2')$ holds. Now putting $g_1 = h_1'$; $g_2 = F_{s_2}h_3'$; $g_3 = F_{s_2s_1}h_2'$; $g_4 = F_{s_2s_1s_3}h_1'$; $g_5 = F_{s_2s_1s_3s_2}h_3'$ and so on, a chain with the desired properties is found. With the aid of the relation 8 of section 4.5 it can be easily shown that $N(g_{k+1},g_k) = 26$. In the table below the first 13 permutations f_k are given.

x	0	1	2	3	4	5	6	7	8	9
$f_1(x)$	3	1	2	0	4	5	6	7	8	9
$f_2(x)$	4	9	1	2	5	6	3	0	7	8
$f_3(x)$	4	8	5	0	9	7	2	6	3	1
$f_4(x)$	1	2	6	9	4	5	7	8	3	0
$f_5(x)$	3	5	0	1	2	8	9	6	7	4
$f_6(x)$	3	8	9	4	7	2	6	1	5	0
$f_7(x)$	2	4	8	6	9	0	7	1	3	5
$f_8(x)$	1	7	9	5	0	3	8	6	4	2
$f_9(x)$	1	8	2	9	5	0	4	7	6	3
$f_{10}(x)$	4	3	2	6	8	7	0	1	5	9
$f_{11}(x)$	2	6	9	8	5	4	1	3	0	7
$f_{12}(x)$	2	8	6	3	4	1	0	9	7	5
$f_{13}(x)$	3	1	2	0	4	5	6	7	8	9

This particular code has a period of 12.

In general a code can be constructed as follows: First select a route in the linking-graph K. Second take g_1 in accordance with the route selected, say $g_1 = L_cR_dF_th_j$, where the j is fixed by the route selected. Then f_1 is free in S, provided that $f_1(1)$ and $f_1(0)$ are

suitable for g_1 and the selected route. If the second g is in the canonical representation $L_a R_b F_s h_i$ and if the matching pair given by the route selection is $F_u h_i, R_e h_j$ then $d \times a = t(e)$ and $s = rtu$ has to hold. Hence only b can be chosen freely. The same holds for each of the following permutations g_k. For each prechosen route in K there are 8! possible choices for f_1, 2000 for g_1 and 10 for each following g_k. Note that the pruned-off links may be used at the ends of the chain. All these codes detect all single errors, all transpositions and all phonetic errors. Of the twin errors 95.5% is detected and of the jump transpositions and jump twin errors 94.2% is detected. In all classes the detection is optimal for codes defined by a check equation $\Pi\ f_i(a_i) = c$ in the dihedral group D_5.

Chapter 5. The bi-quinary codes.

5.0. Introduction.

Historically this chapter should have preceded the foregoing one. In it, a code is explained, which was the first, pure decimal, one to detect all single errors and all transpositions. Like the I.B.M. code mentioned in 2.3, it was designed without regard for the jump errors or the twin errors. It was sheer luck that the first one did detect more than 50% for these types. In 5.3. a generalization is given which scores nearly 90% in said classes. It is very remarkable that the first bi-quinary code is, at the same time, a code based on the dihedral group, whereas the generalization is not interpretable as such. As a matter of fact, the present author tried in 1955 to design a transposition-proof code based on the dihedral group, but without succes. Instead the bi-quinary code of 5.1 was found.

Though the bi-quinary code met the requirements, set at that time, it was considered to be of mainly theoretical interest, since the complexity of the check equations did not encourage the design of a verifier. For a switching circuit, which performs the checking, see 51. The circuit is incorporated in a larger switching system used in the library of the University of Technology at Delft (see 52).

Later on, A. Benard gave an interpretation of the code based on the addition modulo 10. The weights used in the check equation are dependent on the value of the code word itself. It is therefore a non-linear code and for that reason the non-existence proof of 3.2 does not apply. The generalization of the code admitts an analogous interpretation.

5.1. The first bi-quinary code.

The set {0,1,2,3,4,5,6,7,8,9} is mapped on the Cartesian product of the sets {0,1,2,3,4} and {0,1} . The five element set will be denoted by V and the set with two elements by W. The set of the ten decimals is called D. Each decimal digit x is thus mapped on a pair (v,w) with $v \in V$ and $w \in W$. The mapping is quite arbitrary, but it may be advantageous to use a natural way, like $v=x \pmod 5$ and $w=x \pmod 2$. In this chapter the digits 1,2,3,4,5 will be called low and the other ones high. This

is done in accordance with the conventional telephone switching techniques, in which the 0 is represented by 10 pulses and the other digits by the number of pulses indicated by that digit. So, low means less than 6 pulses and high means more than 5 pulses. If a is mapped on (x,y), then it is convenient to have a notation for this relation. Therefore two functions, denoted by v and w, are introduced. The functions map D onto V and W respectively by the definition v(a)=x and w(a)=y. Throughout this chapter only mappings are used such that $\{x \mid w(x)=0\}$ is a complete set of representatives modulo 5. In this section w(x)=0 holds for the low digits and w(x)=1 for the high ones. The sets V and W can be made into groups by defining an addition. For V this addition is the addition modulo 5 and for W it is the addition modulo 2. Hence $(V,+)=C_5$ and $(W,+)=C_2$, since both groups are cyclic. As usual in mathematics, it is not thought necessary to employ different signs for the various additions. For untrained readers and computers this is sometimes confusing. Let $a_1a_2...a_n$ be a word of D^n and let t_j be defined by the recursion $t_{j+1}=t_j+w(a_{j+1})(\text{mod } 2)$ and $t_0=0$, hence $t_j \in W$. The first bi-quinary code C consists of those code words satisfying: $t_n=0$ in C_2 and $(-1)^{t_1}(v(a_1)-v(a_2)-2w(a_1))+(-1)^{t_3}(v(a_3)-v(a_4)-2w(a_3))+... =0$ in C_5. The terms $w(a_i)$ occuring in the latter equation, which are 0 or 1 in W, have to be interpreted as 0 and 1 in V. Strictly speaking a mapping Φ had to be introduced which maps W into V, such that $\Phi(0)=0$ and $\Phi(1)=1$. In the formula, $\Phi(w(a_i))$ should then have been used, instead of $w(a_i)$. The following lemmata can be proved:

5.1.1 The code C is E_1-proof.

5.1.2 $|C|=10^{n-1}$, i.e. the code can be considered as a code with (n-1) information digits and 1 check digit.

5.1.3 The code C is transposition-proof.

5.1.4 The code C is phonetic error-proof.

re 5.1.1 The change of any digit a_i may imply a change of $w(a_i)$, in which case the first check equation ceases to be valid. Otherwise the quinary value $v(a_i)$ has to change, but this will violate the second equation, since all the values $w(a_i)$ and t_j are unaffected by the change.
re 5.1.2 For each of the 10^{n-1} choices of a_i, with $i \geq 2$, it is possible to find one and only one digit a_1, so that both equations are valid.

Through the first equation, $\sum_{i=1}^{n} w(a_i)=0$, the value of $w(a_1)$ is fixed. After that, all the functions t_j have a known value too and hence $v(a_1)$ can be solved from the second equation. The pair $v(a_1), w(a_1)$ fixes the digit a_1.

re 5.1.3 The formal proof of this lemma is labourious since many different cases are considered. It will be left outhere, since the property will be proved later on. It is of interest however to explain the clue of the strange second check equation, which contains two parts, namely: $\sum \pm v(a_i)$ and $-2\sum \pm w(a_{2j-1})$. The first part is a weighted sum modulo 5, with weights dependent on the w-values of the digits. The second part is solely dependent on these w-values; it will be called the binary function of the second check equation.

Now the detection of the transpositions in all the bi-quinary codes of this chapter, is based on the following principles:

The first equation, as a straight sum modulo 2, will never detect a transposition. Let a and b be the transposed digits, then:

1) If $w(a)=w(b)$, and therefore $v(a)-v(b)\neq 0$, then the first part of the second equation, $(\sum \pm v(a_i))$, will change value. The binary function can of course not change.

2) If $w(a)\neq w(b)$, and therefore $v(a)-v(b)$ may take all 5 values of V, then the binary function changes value. The first part of the second equation remains constant in this case. This is necessary, since if it were allowed to change, then for one of the 5 values of $v(a)-v(b)$ the change of the binary function would be compensated.

It is left to the reader to check that the present equations satisfy these principles.

It will be clear from the considerations above, that for the binary function many other possibilities exist, since the only requirement is that it changes value if two adjacent digits with different W-value are interchanged.

re 5.1.4 Since $w(0)=1$ and $w(1)=0$ holds, the phonetic error $1x \to x0$, will spoil the first equation. Also for this argument it is good to take 0 as a high digit.

It will turn out in the course of this chapter, that the twin error detection rate is 5/9 and the jump error detection rate will appear

to be 2/3.

5.2. A recursive definition of the first bi-quinary code.

It is possible to define the same code recursively. Let c_0 be chosen so that $w(c_0)=0$ and $v(c_0)=0$, (hence $c_0=5$ in the convention of this chapter). Define c_{2k+1} and c_{2k} by: $w(c_{2k})=w(c_{2k-1})+w(a_{2k})$; $w(c_{2k+1})=w(c_{2k})+w(a_{2k+1})$ and $v(c_{2k})=v(c_{2k-1})-(-1)^{w(c_{2k-1})}v(a_{2k})$; $v(c_{2k+1})=v(c_{2k})+(-1)^{w(c_{2k+1})}(v(a_{2k+1})-2w(a_{2k+1}))$.

The code C consits of those words, for which $c_n=c_0$ holds. From these recursive formulae, which are immediately clear from the equations of the preceding section, it follows that a Latin staircase is possible (see chapter I).

The following two Latin squares are applied alternatively:

	5	1	2	3	4	0	6	7	8	9
5	5	1	2	3	4	7	6	0	9	8
1	1	2	3	4	5	8	7	6	0	9
2	2	3	4	5	1	9	8	7	6	0
3	3	4	5	1	2	0	9	8	7	6
4	4	5	1	2	3	6	0	9	8	7
0	0	9	8	7	6	3	4	5	1	2
6	6	0	9	8	7	4	5	1	2	3
7	7	6	0	9	8	5	1	2	3	4
8	8	7	6	0	9	1	2	3	4	5
9	9	8	7	6	0	2	3	4	5	1

	5	1	2	3	4	0	6	7	8	9
5	5	4	3	2	1	0	9	8	7	6
1	1	5	4	3	2	6	0	9	8	7
2	2	1	5	4	3	7	6	0	9	8
3	3	2	1	5	4	8	7	6	0	9
4	4	3	2	1	5	9	8	7	6	0
0	0	6	7	8	9	5	1	2	3	4
6	6	7	8	9	0	1	2	3	4	5
7	7	8	9	0	6	2	3	4	5	1
8	8	9	0	6	7	3	4	5	1	2
9	9	0	6	7	8	4	5	1	2	3

The entries of the two tables are written in a somewhat unusual order to show the similarity with the multiplication table of the dihedral group. In fact, after interchanging 0 and 5 both in the entries and in the body of the table, two tables are obtained, which are column permutations of the Cayley table of D_5 on page 83. Let the dihedral group after the interchange of 0 and 5, be denoted by D_5', hence in D_5' $5=\delta^0$; $0=\varepsilon$ and $j=\delta^j$; $j+5=\delta^j\varepsilon$, for j=1,2,3,4. The recurring relations become:

$c_{2k+1}=c_{2k}\times f_1(a_{2k+1})$ and $c_{2k}=c_{2k-1}\times f_2(a_{2k})$ in D_5', where using the conventions of chapter 4, $f_1(\delta^i)=\delta^i;f_1(\delta^j\varepsilon)=\delta^{2-j}\varepsilon$ and $f_2(\delta^i)=\delta^{-i};f_2(\delta^j\varepsilon)=\delta^{-j}\varepsilon$. Thus $g_1=f_1f_2^{-1}(\delta^i)=\delta^{-i};g_1=f_1f_2^{-1}(\delta^j\varepsilon)=\delta^{j+2}\varepsilon$ and $g_2=f_2f_1^{-1}(\delta^i)=\delta^{-i};$ $g_2=f_1f_2^{-1}(\delta^j\varepsilon)=\delta^{j+3}\varepsilon$.

Hence both g_1 and g_2 are permutations of the type given in the example of section 4.3, which gives rise to a transposition-proof code. The code can thus also be defined by $\prod_{k=1}^{n} h_k(a_k)=5$ in D_5', with $h_{2k+1}=f_1$ and $h_{2k}=f_2$. From this interpretation of the code it follows immediately that the detection rate of the twin errors is 5/9, see p. 91. The jump error detection rate is also easily found by the method of 4.2. In order to apply this method the distribution of the pairs $(x,g_1g_2(x))$ and $(x,g_2g_1(x))$ over the classes A_{ij}, should be known. Now g_1 and g_2 are each others inverse, so that all the pairs fall in A_{00} and A_{11}, that is 5 in each class. Moreover, the difference of the exponent of δ in x and $g_1g_2(x)$ is always 0. Hence there are 20 pairs of x,z which cause 5 undetected jump errors each. To the complementary classes belong 25 pairs x,z each giving 2 undetected jump errors, so that only 300 of the 450 possible jump errors will be detected.

It is the tragedy of codes like the one above, that even though the error-type with the low detection rate, has a small frequency, it may occur that in the set of undetected errors, the given type is dominant. The result is that the code looks very bad, giving the impression that a major class of errors has been overlooked.

By taking another mapping of D on V×W, A. Benard gave an elegant interpretation of the same code.

Let the mapping be defined by v(x)=x(mod 5) and w(x)=x(mod 2). Let $a_1a_2...a_n$ be a code word. Now Benard remarks that the odd digits separate the even digits in, possibly empty, runs. The runs, including the empty ones can be numbered from the left to the right A run with an even serial number, will be called an even run. Let the odd digits of the code word be numbered from the left to the right and let o(j) denote the j-th odd digit. Let furthermore the even digits of the code word also be numbered from the left to the right and let e(j) denote the j-th even digit. Now put $\sum(-1)^j o(j)=S_o$ and $\sum(-1)^j e(j)=S_e$, and let the

number of even digits, occurring in even runs, be denoted by K. The code C is defined as the set of code words satisfying: $S_o-S_e=2K$ (mod 10). This code is essentially the same as the one defined in 5.1. The check equation $S_o-S_e=2K$ taken modulo 2 is the same as the first equation of 5.1 and taken modulo 5 the equation becomes: $\sum(-1)^j v(o(j))-\sum(-1)^j v(e(j))=2K$ (mod 5).

Now let a_i be the j-th odd digit, then $a_i=o(j)$ and $t_i=j$ (mod 2). If $i=2i'-1$ then $j=t_{2i'-1}$ (mod 2) and if $i=2i'$ then $j=t_{2i'}=t_{2i'-1}+w(a_i)=t_{2i'-1}+1$(mod 2).

Hence for the odd digits the coefficient of $v(a_i)$ is the same in the modulo 5 equation of both interpretations. On the other hand, if a_i is the j-th even digit, then $a_i=e(j)$ and $j=i-t_i$(mod 2). Hence for $i=2i'-1$ it holds that $j=1+t_{2i'-1}$(mod 2) and for $i=2i'$ it follows that $j=t_{2i'}=t_{2i'-1}+w(a_i)=t_{2i'-1}$(mod 2). Hence also for the even digits the coefficients of $v(a_i)$ are the same in both modulo 5 equations. Only the binary function of the Benard equation is different, but 2K does have the property that it changes value if two adjacent digits with different parity are interchanged. It follows that the code defined in the Benard fashion has the same detection rate for the transpositions and the phonetic errors. This can also very easily be proved directly, using the principles of page 104.For interchanging adjacent digits with the same parity, is detected since either S_o or S_e but not 2K changes value. Interchanging digits with different parity only changes 2K, since the even and the odd digits retain their serial number, but one of the even digits comes in a run of a different parity. For the proof that the single errors are detected, it is sufficient to observe that the number of odd digits in a valid code word is always even. A parity changing single error disturbs this rule and a non-parity-changing error is detected by S_o or S_e, since all the coefficients are unaffected by the error.

The binary function, 2K, in the Benard variant is a less fortunate choice, since the twin errors are only detected if even twins from an even run are changed into odd twins. This gives a twin error detection rate of about (25/45)/2=27.8%. The jump error detection is independent of the binary function.It should be noted that parity may be read as W-value.

5.3 Generalization of the bi-quinary code.

The purpose of generalizing a formula is often to create the possibility

of selecting another specialization, which has more desirable properties than the original formula. In other words after creating more freedom of choice the selection of a better code becomes feasable. It is a delicate question what a proper generalization is in this respect. The Latin staircase method with arbitrary Latin squares, for instance, is certainly a generalization, but it is of little help because there is no easy way to test the merits of the resulting code combined with an overwhelming number of possibilities. A generalization should preserve some basic idea. Finding and formulating the basic idea of a method is essential for finding a generalization. The clue of the present code is thought to be the bi-quinary representation of the decimals in combination with the peculiar structure of the second check equation. The success hinges on the fact that a quinary transposition-proof code is possible. A weighted quinary code, defined by $\sum u_k a_k = o \pmod 5$, is transposition-proof if the adjacent weights are different. These weights may depend on the binary components of the digits. The binary word $w(a_1)w(a_2)\ldots w(a_n)$ will be called the binary key of the decimal code word $a_1 a_2 \ldots a_n$. The binary key should be parity checked and therefore it detects always the single errors which change the binary key. Those single errors which leave the binary key invariant are to be detected by the quinary check equation $\sum u_i v(a_i) = B \pmod 5$, where B is the key dependent binary function. This implies that none of the coefficients u_i may vanish. The principles of the transposition detection were (see page 104): that the left hand side of the quinary equation remained invariant if the binary key changed and was changed if the binary key remained invariant. The latter property is fulfilled as soon as the adjacent weights are different. The invariance under key changing transpositions is a much more severe requirement for the key dependent weights u_i. Let b_1 and b_2 be two keys, which are equal on all places but the first two and let b_1 start with 01 and b_2 with 10, then $u_i(b_1)=u_i(b_2)$ for $i > 2$ has to hold. Furthermore $u_1(b_1)=u_2(b_2)$ and $u_1(b_2)=u_2(b_1)$ are also necessary conditions. For each key b, it has to hold that $u_i(b) \neq u_{i+1}(b)$, if b has equal bits on the i-th and the (i+1)-th position.

The obvious improvement strived for is a better detection of the twin errors and the jump transpositions. A cumbersome analysis reveals that

100% detection cannot be achieved in either category by this method. The optimal result can be explained best by reconsidering the first bi-quinary code. The coefficients occurring in the second check equation are exclusively +1 or -1, so that an alternating quinary check is employed. It is however well-known that equations like $\sum 2^i a_i = 0 \pmod 5$, yield much better codes for pure quinary code words. It is therefore obvious to try to exploit this circumstance. A decimal code can be defined as follows:
Define, using the same notation as in 5.2, T_o and T_e by $T_o = \sum 2^j o(j)$ and $T_e = \sum 2^j e(j)$ and let B be a binary function, which is modulo 5 sensitive for the transposition of digits with unequal parity (or W-value). The code C is defined as the set of words satisfying: $\sum w(a_i) = 0 \pmod 2$ and $T_o + T_e = B \pmod 5$. If w is defined by $w(x) = x \pmod 2$ then the two equations may be combined into one by setting $T_o' = \sum 7^j o(j)$, giving $T_o' + T_e = B' \pmod{10}$, where B'=B if B is even and B'=B+5 if B is odd. The equation modulo 5, just as the second equation of the first bi-quinary code, has the property that the interchange of adjacent digits, with different parity, does not change the left hand side, since there is no change in the serial number of the odd or even digits. The binary function B however will change. On the other hand, if two digits with the same parity are interchanged, then the function B will not change, whereas one of the sums T_o or T_e will. Hence each transposition will disturb the second check equation. The advantage of the generalization is that the non-parity-changing twin errors are always detected. This follows at once from: $2^j a + 2^{j+1} a = 2^j (3a) \neq 2^j (3a') = 2^j a' + 2^{j+1} a'$ and $a - a' \neq 5$. The parity-changing twin errors disturb a lot in the equation, since all odd and all even digits, which follow the error, get an other serial number. Also the function B may change value. For each a there are five possible values for a', which are all different modulo 5, hence for each a there is just one a' which compensates whatever changes occurred through the parity change. There are never two values for a' which do so, since otherwise the single error which interchanges these two a''s would not be detected. So 5 of the 45 twin errors per position, are undetected, giving a rate of 8/9
For the jump errors there are several cases to be considered.

i) The interchanged digits and the digit in between, all have the same parity. This type is detected, since $2^j a + 2^{j+2} b \neq 2^j b + 2^{j+2} a$ is equivalent with $2^j (3a) \neq 2^j (3b)$ and since $a - b \neq 0$.

ii) The interchanged digits have the same parity, which is different from the parity of the middle one. These errors are detected since $2^j a+2^{j+1}b \neq 2^j b+2^{j+1}a$ holds, as $a \neq b$.

iii) The interchanged errors have a different parity. Now 5 out of the 25 possible combinations are undetected, since for each a there are 5 different values for b possible. Only one of these values will leave the second equation true. The nett result is that the jump transposition detection rate is 8/9. The jump twin errors are more slippery. Suppose that aba becomes cbc somewhere in the code word. Consider three cases.

i) $w(a)=w(b)=w(c)$. Then the error is not detected because $2^j a+2^{j+2}a=2^j(5a)=0 \pmod 5$.

ii) $w(a)=w(c) \neq w(b)$. This error is detected because $2^j a+2^{j+1}a=2^j(3a) \neq 2^j(3c)=2^j c+2^{j+1}c$ and since $a \neq c \pmod 5$.

iii) $w(a) \neq w(c)$. Then again 1 of the 5 possible values of c gives a valid check equation.

Hence 15 of the 45 errors will be undetected, thus yielding a detection rate of 2/3. This latter rate is not easily improved upon.

The choice of the function B is less important in this generalized code, but for some technical implementations a skrewed choice may be of influence. To make this clear an example is sketched. Let e_j be the number of even digits preceding the j-th digit of a code word and let o_j be the number of odd digits preceding the j-th one. Hence $e_j+o_j=j-1$. Now $B=\sum_{w(a_j)=1}(2^{o_j}-2^{e_j})$ may be taken and the second check equation may be written as $\sum_{w(a_j)=1} 2^{o_j}a_j+\sum_{w(a_j)=0} 2^{e_j}a_j=B \pmod 5$. Let

v and w again be defined by $w(x)=0$ for $x \in \{1,2,3,4,5\}$ and else $w(x)=1$, and $v(x)=x \pmod 5$. Suppose that the digits are fed into a verifier, as pulse trains according to the convention that the digit x is represented by a train with x pulses, with the understanding that the 0 is counted for 10. Without going into the details, it may be pointed out that the high digits are recognized only after the 6-th pulse is received. It can be so arranged that the pulses of each train are treated in the beginning according to the "even mode" and only after receiving 6 pulses the treatment is changed into the "odd mode". The

result is that the even (that is low) digits are counted correctly, but that the high digits gave 6 pulses in the wrong mode, whereas a_j-6 pulses are treated correctly. This difference modulo 5 is precisely needed for the binary function B. The peculiar form of the function used in 5.1 also comes from technical considerations.

Though this generalized bi-quinary code improves upon the one of 5.1, it is still of a lower standard than the codes of chapter 4. Mathematically it has a very interesting property, which alone would be a sufficient reason, or excuse, for mentioning it. It is the only code so far which is not of the Latin staircase type. In other words, there is no recursive definition like $c_{i+1}=\phi_i(c_i)$.

The recursion can be given with the aid of an auxiliary binary quantity ε_i. From a technical point of view this means that an extra memory is needed.

1. Beckley, D.F.
Check digit verification.
Data Processing, 1966, p. 194-201

2. Beckley, D.F.
An optimum system with 'modulus 11'.
The Computer Bulletin, 1967, p. 213-215

3. Bell telephone manufacturing company
Rekeninrichting voor het berekenen van een aantal cijfers die tezamen een getal vormen.
Nederlands octrooi, No. 92399, 1959

4. Bell telephone manufacturing company
Rekeninrichting voor het berekenen van een lineaire cijferfunctie van een aantal cijfers die tezamen een getal vormen.
Nederlands octrooi, No. 92330, 1959

5. Berlekamp, E.R.
Algebraic Coding Theory.
Mc Graw-Hill Book Company, 1968.

6. Bijleveld, W.J.
Fouten voorkomen of controle cijfers?
Informatie, 1967, vol. 9/11, p. 239-242

7. Bourland, D.
Non-decision theory.
Datamation, 1964, p. 52-53

8. Carmichael, R.D.
Introduction to the theory of groups of finite order.
Reprint Dover Publications Inc. 1956

9. Chien, R.T.
Correction coding apparatus for decimal-like codes.
IBM Technical Disclosure Bulletin, 1965, vol.7, p. 781-784

10. Compagnie des machines Bull
Inrichting voor het bepalen van een controlesymbool behorende bij een decimaal getal.
Nederlands octrooi, No. 81419, 1956

11. Corput, J.G. van der
A remarkable family.
Euclides, 1941, vol. 18, p. 50-78

12. Eberlein, G.
Die automatische Nummerprüfung.
Handbuch der machinellen Datenverarbeitung, 1965, vol. 1/4/3

13. Friedman, W. and Mendelsohn, C.J.
Notes on codewords.
Am. Math. Monthly, 1932, vol. 39, p. 394-409

14. Garrison, G.N.
Quasi Groups.
Annals of Mathematics, 1940, vol. 41(3) July, p. 474-487

15. Golomb, S.W.
Polyominoes.
C. Scribner's Sons, New York, 1965

16. Hall, jr, M.
The theory of groups.
The Macmillan Company, New York, 1959

17. Hamming, R.W.
Error detecting and error correcting codes.
The Bell System Technical Journal, 1950, vol. 26, p. 147-160

18. Herger, H.
Das Anker-zahlenprüfgerat F 1300.
1966, p. 1-27

19. Huffman, D.A.
A method for the construction of minimum-redundancy codes.
Proceedings of the I.R.E. sept. 1952, p. 1098-1101

20. I.B.M.
Ponsinrichting.
Nederlands octrooi, No. 93551, 1960

21. I.B.M Form 71 204
Nummerprüfung für Kartenlocher IBM 24 und 26. (Ubersetzung und Bearbeitung der Broschure Self-checking number device for IBM 24-26 card punches. Form No. 22-6022-2, 1963)

22. I.B.M. Form 79 953
Prüfziffernverfahren, Belegverarbeitung Praxis Nr 3.
Matt, G. und Jülicher, W. 1964

23. International Standard Electric Corp.
Rechenanordnung zur Berechnung einer Prüfziffer aus einer Dezimalzahl.
DAS 1 187 831 42m-14 vom 25-2-1965 (14.9.1961)

24. Joshi, D.D.
A note on upper bounds for minimum distance codes.
Information and control, 1958, vol. 1, p. 289-295.

25. Kahn, D.
The Codebreakers.
The story of secret writing.
The Macmillan Company, New York, 1967

26. ---
Kontrolle mit Hilfe von Kontrollziffern.
Bürotechn. Sammlung, September 1956

27. Kraitchik, M.
Mathematical recreations.
second edition,
Dover Publications Inc. 1953

28. Lauter, F.
Der Wert und die Notwendigkeit der automatischen Nummercontrole in Datenverarbeitungsanlagen.
Rat. Büro, 1964, vol. 5, p. 281-283.

29. Lee, C.Y.
Some properties of Nonbinary Error-Correcting Codes.
IRE Transactions on Information Theory, 1958, vol. 4, p. 77-82

30. McRae, T.W.
Self-checking number codes, an aid to accurate accounting reports.
The Accountant, 1964, vol.50, p. 611-614

31. Moll, W. de
Beschouwing omtrent het gebruik van controle-cijfers.
Informatie, 1962, Nr. 19, p. 8-9

32. Nasvytis, A.
Die Gesetzmäszigkeiten kombinatorischer Technik.
Springer Verlag, Berlin Göttingen Heidelberg, 1953

33. Nederlandse Spoorwegen
De nieuwe kenmerken op goederenwagens.
1965

34. Oberman, R.M.M.
A method of reversible cyphering of administrative numbers.
Proc. Symposium on automation of population register systems, Jerusalem, 1967, p. 455-462

35. Ore, O.
Theory of graphs.
American Mathematical Society
Colloquium Publications, 1962, vol. 38

36. Owsowitz, S. and Sweetland, A.
Factors affecting coding errors.
Memorandum, 1965, RM-4346-PR
The Rand Corporation , Santa Monica, California.

37. Peterson, W.W.
Error-correcting codes.
The M.I.T. Press and John Wiley and sons Inc. 1961

38. Plotkin, M.
Binary codes with specified minimum distance.
IRE Transactions on information theory, sept. 1960, p. 445-450

39. Renelt, G. en Schröder, J.
Beveiliging van informatie bij invoer en transmissie met behulp van een of twee controletekens.
Philips Technisch Tijdschrift, 1965, vol. 26, p. 323-331.

40. Richards, D.L.
The incidence errors in dialling.
Teleteknik ,1963, p. 53-58.

41. Rothert, G.
Influence of dials and push-button sets on errors, including the time required for the trasnmission of numbers.
Teleteknik, 1963, p. 59-66

42. Ryser, H.J.
Combinatorial Mathematics.
The Carus Mathematical Monographs 14.
Wiley and sons, 1963

43. Schauffler, R.
Uber die Bildung von Codewörtern.
Archiv der Electrischen Ubertragung, 1956, vol. 10, p. 303-314.

44. Selmer, E.S.
Registration numbers in Norway;
Some applied number theory and psychology.
Informatie, 1967, vol. 9, p. 167-170

45. Shapiro, H.S. and Slotnick, D.L.
On the mathematical theory of Error-correcting codes.
I.B.M. Journal of Research and Development, jan. 1959, vol. 3 No. 1, p. 25-34

46. Sisson, R.L.
An improved decimal redundancy check.
Communications of the association for Computing Machinery, 1958, vol. 1, p. 10-12.

47. Steeneck, R.
Error Detection, Correction and Control.
Western Union Technical Review, 1962, vol. 16/3, p. 134-139

48. Tunis, C.J
A decimal error correcting technique.
IBM Technical Disclosure Bulletin, 1963, vol. 5, p. 42-43

49. Ulrich, W.
Non-binary error correcting codes.
Bell System Technical Journal, 1957, vol. 36/6, p. 1341-1388

50. Verhoeff, J.
Trends in Library Building
Libri 1965: vol.15: no.1: pp. 56-61

51. Verhoeff, J.
Inrichting voor het controleren van een symboolgroep, waaraan een controlegetal is toegevoegd respectievelijk voor het bepalen van dit controlesymbool.
Nederlands octrooi aanvrage 6400631, 1964

52. Verhoeff, J.
The Delft Circulation System.
Libri, 1966, vol. 16, p. 1-9

53. Verhoeff, J.
Fout-ontdekkende en corrigerende codes.
Kantoor en efficiency, 1966, nr. 47, p. 1976-1980

54. Verhoeff, J.
Error detecting and correcting codes for the decimal numbersystem.
Proc. Symposium on automation of population register systems, Jerusalem, 1967, p. 447-454

55. Zassenhaus, H.
Lehrbuch der Gruppentheorie.
English translation: The theory of groups.
Chelsea Publishing Company, New York, 1949